Review on Variational Autoencoders

Jannie Yu

January 2020

1 Introduction

Recent advances in biological experimentation methods allow us to draw massive quantities of data from different domains. These rich datasets make these problems particularly well-suited to techniques like deep learning, allowing us to better quantify cellular processes. However, the complexity of biological processes poses unique challenges to researchers. Biological data consists of multiple nonlinear and redundant connections among processes, making learning incredibly difficult. There is also a lack of ground truth labels in datasets, inhibiting the power of supervised learning. A new class of unsupervised learning models such as variational autoencoders (VAE) and generative adversarial nets (GAN) have shown promise in learning through data reconstruction without the need of labels. These models allow for latent space interpolation, which enables biologists to capture a constrained, lower dimensional space for the distribution of data. In turn, one can use the reconstructed data to reveal novel biological patterns in various domains. This paper serves as an in-depth analytical review of VAEs. Other exploratory data analysis techniques like principal component analysis (PCA) will also be briefly covered.

2 Exploratory Data Analysis Techniques

2.1 Principal Component Analysis (PCA)

Principal component analysis is an algorithm that conducts dimension reduction. By reducing the dimension of the feature space, one can have fewer relationships between variables to visualize their distance and relatedness [1]. This technique reduces the possibility of overfitting the data by removing less relevant features in our analysis. Thus, given a data set X of dimension p, the goal is to construct a data set Y of smaller dimension L.

2.1.1 Algorithm Outline

The algorithm works as follows:

(1) Set up the $n \times p$ data matrix X

(2) Using X, construct a centered data matrix Z. This is done by subtracting each column's element from the column's mean.

(3) Find the $p \times p$ covariance matrix $\frac{Z^T Z}{n-1}$ of Z

(4) Calculate the eigenvectors and the corresponding eigenvalues of the covariance matrix by using eigendecomposition. The matrix will be decomposed into VDV^{-1}, where V is the matrix of eigenvectors and D is the diagonal matrix with eigenvalues on the diagonal.

(5) Sort the columns of the eigenvector matrix V and eigenvalue matrix D in order of decreasing eigenvalue.

(6) Select the first l columns of the sorted V as the columns of a new $p \times l$ matrix W (You can find the best l through computation of the cumulative energy content for each eigenvector, or preset it to be

the number of features that you want to retain).

(7) Find the new projected vectors of the new basis in the columns of the matrix $Z\dot{W}$.

An example of the PCA algorithm implementation is at the back of the review paper; this example is a MATLAB implementation done for ACM 104 Applied Linear Algebra in Fall 2019.

2.1.2 Detailed Explanation of Algorithm

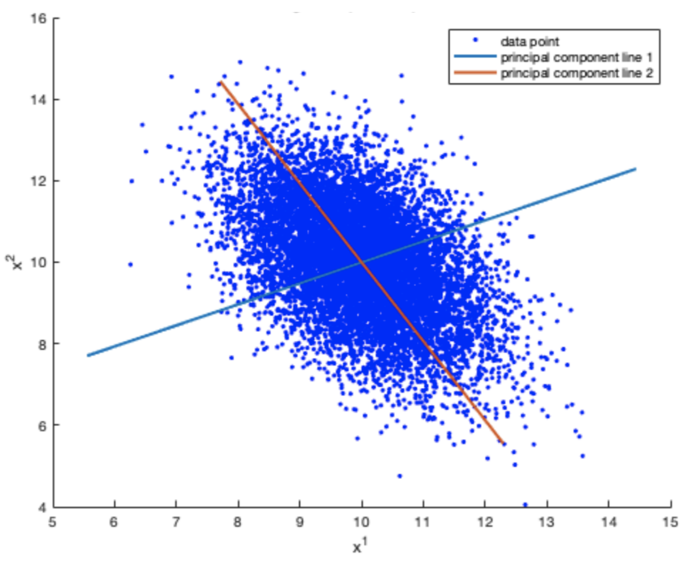
Suppose that we are trying to understand some phenomenon in a biological system. In this system, we are able to quantitatively measure various quantities like gene expression, velocity, etc. Let's suppose that during data collection, we obtain values for the n unique quantities for each of our p cell measurements. Let's first assemble all of this data into a data matrix, where the row x^i is the values of the i-th quantity across all measurements and the column x_j is the j-th measurement. Thus, an element x_{ij} in the matrix is the value of the i-th quantity in the j-th measurement.

Now, we want to find the centered data matrix Z. This is done by subtracting each column's element from the column's mean. This centers the matrix so that the mean of each column is zero. One can further standardize the matrix; this can be done if the variance of features is independent of the importance of a feature. To standardize, divide each column's element by the column's standard deviation.

Next, we want to get the covariance matrix $\frac{Z^T Z}{n-1}$. This matrix gives the covariance between each pair of elements. The covariance is positive if two values have a direct relationship. The covariance is negative if two values have an inverse relationship. In simple terms, it contains estimates of how variables in Z relate to one another.

From there, use eigendecomposition of the covariance matrix to calculate the eigenvectors and the corresponding eigenvalues. The matrix will be decomposed into VDV^{-1}, where V is the matrix of eigenvectors and D is the diagonal matrix with eigenvalues on the diagonal. The eigenvectors will form a new basis for your data, where each individual eigenvector are axes along which linear transformation acts, stretching or compressing input vectors. The larger the corresponding eigenvalue, the stronger the axes magnitude, or in other words, the measure of the data's covariance. We want to have principal component lines that fit strongly to the variance of the data, so by ordering the columns of the eigenvector matrix V and eigenvalue matrix D in order of decreasing eigenvalue, you get the principal components in order of significance. After ordering, we select the first l eigenvectors from the reordered eigenvector matrix and have then be columns of a new matrix W. l can be predefined as a lower dimension $l \in (1, n)$, or be calculated using

Finally, we want to reorient the data with a change of basis. A useful property of eigenvectors is that the vectors are orthogonal to each other. Thus, we can use a new coordinate system in a new space defined by its lines of greatest variance represented by the principal components. The projected vectors will be the columns of the matrix $T = Z\dot{W}$.



Example of principal component lines as axes accounting for the most variance in the data

PCA is helpful because it decomposes matrices in ways that show us the most important features that are not obvious from the representation of the matrix as an array of elements. The principal components can be thought as axes that accounts for the most variance, so the first component will be along the dimension that reduces unpredictability the most, or most decrease in system's entropy. Thus, they perform the same task as the autoencoders employed by deep neural networks, with one primary difference; PCA is restricted to a linear map while autoencoders can have nonlinear mappings. Thus, the PCA technique is equivalent to a single-layer autoencoder with a linear transfer function.

3 Architecture of VAEs

Variational Autoencoders (VAE) are generative models, similar to Generative Adversarial Networks (GAN). In a generative model, one models the conditional probability of the observable X, given a target y. VAEs are architecturally similar to GANs in that both use autoencoders. However, they have different purposes and mathematical formulation. GANs have the explicit goal of optimizing for generative tasks. Meanwhile, VAEs are directed probabilistic graphical models with an explicit goal of latent modeling. As a result, the model makes certain assumptions about the distribution of latent variables, allowing the latent space to be continuous. This in turn allows the model to make easy random sampling and interpolation from the latent space.

3.1 Mathmatical Formulation

The goal of a VAE is to find to infer good values of the latent variables given observed data. Let's define a probability model for a dataset and latent variables z.

To write the goal in formal terms, we want to find the posterior $p(z|x)$. Bayes' Law defines the posterior probability as

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

where $p(x|z)$ is the probabilistic likelihood of a data x appearing given the latent variable, $p(z)$ is the prior belief of the probability distribution function of the latent variables, and $p(x)$ is the probability distribution function of the "evidence", or x. We can calculate $p(x)$ by margining out the latent variables to get

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

The integral needs to be evaluated over all possible configurations of latent variables, so it would take exponential time to compute. As a result, one can use neural networks to learn the best approximation $q(z|x)$.

To figure out how much $q(z|x)$ approximates the true $p(z|x)$, we can use the Kullback-Leibler divergence. The KL divergence measures how different one probability distribution is from another reference probability distribution. To do this, it calculates the expectation of the difference of information content for the two distributions. It can also be described as a measurement of surprise. Let's break down the derivation for the divergence:

The information content of an event is the amount of "surprise" that comes from it. For example, if an event occurred and had a high probability of happening, there is low "surprise". Thus, the information content of an event x with respect to distribution p is

$$I_p(x) = -\log p(x)$$

Next, the difference in information content between distribution p and q is

$$\Delta I = I_p - I_q = -\log p(x) + \log q(x) = \log(\frac{q(x)}{p(x)})$$

The KL divergence is the expectation of the difference, and thus is given by

$$D_{KL}(q(x)||p(x)) = E\ _q[\Delta I] = \int(\Delta I)q(x)dx = \int q(x)\log(\frac{q(x)}{p(x)})dx$$

To get our optimal approximate posterior, we want to find a posterior such that the KL divergence between our approximated $q(z|x)$ and true $p(z|x)$ is minimized. Thus, our goal is to minimize

$$\arg\min_{\lambda} \mathbb{KL}(q_\lambda(z|x)\ ||\ \ p(z|x))$$

$$= \mathbb{E}_q[\log q_\lambda(z|x)] - \mathbb{E}_q[\log p(x,z)] + \log p(x)$$

Again, $p(x)$ takes an exponential time to evaluate, so instead of directly analytically evaluating the KL divergence, we need to find an approximate form of KL divergence to optimize. If we break up the KL divergence further, we obtain:

$$\mathbb{KL}(q(z|x)\ ||\ \ p(z|x)) = E_{Z\sim q}[\log \frac{q(Z)}{p(Z|x)}]$$

$$= E_{Z\sim q}[\log q(Z)] - E_{Z\sim q}[\log p(Z|x)]$$

$$= E_{Z\sim q}[\log q(Z)] - E_{Z\sim q}[\log p(Z,x)] + E_{Z\sim q}[\log q(Z)]$$

$$= \log p(x) - (E_{Z\sim q}[\log p(x,Z)] - E_{Z\sim q}[\log q(Z)])$$

$$= \log p(x) - \text{ELBO}$$

where

$$\text{ELBO} = (E_{Z\sim q}[\log p(x,Z)] - E_{Z\sim q}[\log q(Z)])$$

Thus, by maximizing ELBO, or the evidence lower bound, one effectively is maximizing the lower bound of the probability of generating real data samples. This allowed us to minimize the KL divergence without access to $p(x)$.

At the same time, we want to maximize the likelihood that the latent variables found produces the data that was actually observed. Thus, we want to maximize the expected value of $\log p_\theta(x|z)$ with respect to the latent space representation $z \sim q(z|x)$. This can be written as $\mathbb{E}_{q_\phi(z|x)}(\log p_\theta(x|z))$.

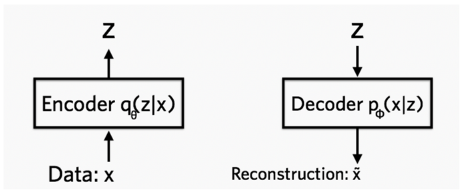
The final objective function that we want to minimize is

$$\mathcal{L}(\phi, \theta, x) = \mathbb{KL}(q_\lambda(z|x) \;||\; p(z)) - \mathbb{E}_{q_\phi(z|x)}(\log p_\theta(x|z))$$

$$\mathcal{L}(\phi, \theta, x) = -\text{ELBO} - \mathbb{E}_{q_\phi(z|x)}(\log p_\theta(x|z))$$

3.2 Neural Network Architecture

3.2.1 Optimizing the Objective



The VAE model consists of two neural networks: an encoder and a decoder. The purpose of the encoder is to map its input, a data point x to a point z in the latent space following an approximated posterior probability distribution $q_\phi(z|x)$, where ϕ represents the parameters of the encoder. We want this distribution to approximate $p(z)$ as close as possible, so the encoder updates by finding parameters that minimize the KL divergence between $q_\phi(z|x)$ and $p(z)$. The purpose of the decoder is to map a latent space variable z to to a reconstruction $\tilde{x}$ following the likelihood probability $p_\phi(x|z)$, where ϕ represents the parameters of the decoder. Since we want this likelihood to be maximized, the decoder updates by finding parameters that maximises the log-likelihood. Thus, by updating the encoder and decoder, one effectively minimizes the objective function for the VAE model.

3.2.2 Reparameterization Trick

Since we created a probablistic model, the output of the encoder will be non-deterministic. As a result, we cannot simply use gradient descent to update the autoencoders; the gradient of a model is how the output changes when the parameters are changed, but the VAE model produces an output is different with each run even without altering the parameters. As a result, we cannot optimize the model directly, but we can optimize the most probable output of the model. Knowing that we can describe the output of the model using normal distribution, where x is the input and σ is the standard deviation parameter in $\mathcal{N}(x, \sigma^2)$, we can use a reparameterization trick to be able to update and perform backpropagation in the right direction:

(1) Sample noise variable ϵ from a simple distribution $p(\epsilon)$ like the standard Normal $\mathcal{N}(0, 1)$. In other words, sample an auxiliary independent random variable.

$$\epsilon \sim p(\epsilon)$$

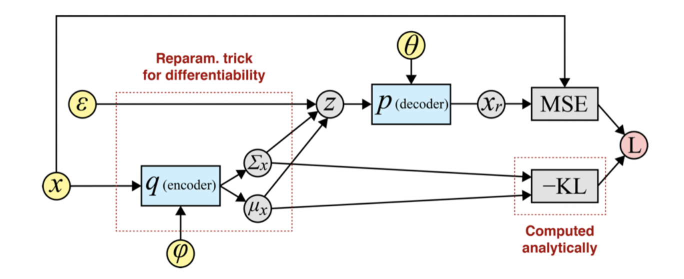
(2) Apply a deterministic transformation that maps the random noise into a more complex distribution.

$$z = g_\phi(\epsilon, x)$$

$$z = g_{\mu,\sigma}(\epsilon) = \mu + \epsilon\dot{\sigma}$$

This process allows us to take the gradient with respect to $q(z)$ and independent of the variational parameters.

3.2.3 Neural Network Implementation



The autoencoder implementation works as follows:
(1) Push a datapoint x through the encoder
(2) Sample noise $\epsilon \in p(\epsilon)$
(3) Reparameterize using the encoder's parameters ϕ to generate samples from $z \sim q_\phi(z|x)$
(4) Push the sample z through the decoder
(5) Compute reconstruction loss
(6) Compute variational loss with KL divergence (computed analytically)
(7) Combine losses

References

[1] Jonathon Shlens. A tutorial on principal component analysis, 2014.

[2] Tutorial - what is a variational autoencoder?

[3] Lilian Weng. From autoencoder to beta-vae. *lilianweng.github.io/lil-log*, 2018.

[4] Carl Doersch. Tutorial on variational autoencoders, 2016.

[5] Karren D. Yang and Caroline Uhler. Multi-domain translation by learning uncoupled autoencoders, 2019.

[6] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.