EE 046202 - Technion - Unsupervised Learning & Data Analysis

- Formerly 046193

Tal Daniel

Tutorial 09 - Deep Unsupervised Learning - Variational Autoencoder (VAE) - Part 1

Agenda

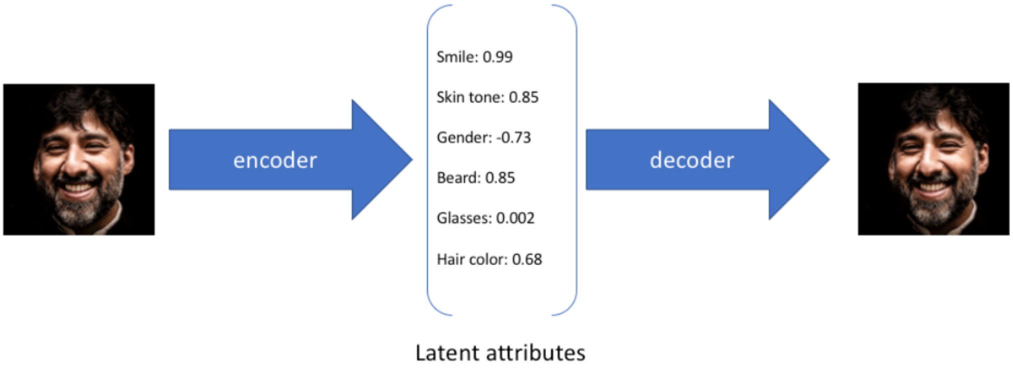
- Motivation and Introduction
  - Autoencoders
  - Explicit and Implicit Generative Models
- Variational Autoencoders (VAE)
  - Evidence Lower BOund (ELBO)
  - VAE's Objective Closed-form Solution
  - Reparameterization Trick

Motivation & Introduction

- Most of the natural data is high-dimensional, such as images. Consider the MNIST (hand-written digits) dataset, where each image has $28x28 = 784$ pixels, which means it can be represented by a vector of length 784.
  - But do we really need 784 values to represent a digit? The answer is no. We belive that the data lies on a low-dimensional space which is enough to describe the observartions. In case of MNIST, we can choose to represent digits as one-hot vectors, which means we only need 10 dimensions. So we can **encode** high-dimensional observations in a low-dimensional space.
  - But how can we learn meaningful low-deminsional representations? The general idea is to reconstruct or, **decode** the low-dimensional representation to the high-dimensional reperesentation, and use the reconstruction error to find the best representations (using the gradients of the error) . This is the core idea behind **autoencoders**.

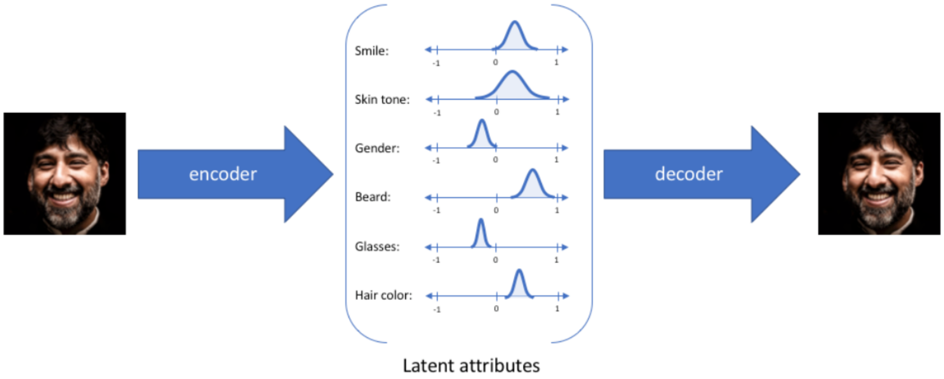
- **Autoencoders** - models which take data as input and discover some latent state representation of that data.The input data is converted into an encoding vector where each dimension represents some learned attribute about the data. The most important detail to grasp here is that our encoder network is outputting a single value for each encoding dimension. The decoder network then subsequently takes these values and attempts to recreate the original input. Autoencoders have **three parts**: an encoder, a decoder, and a 'loss' function that maps one to the other. For the simplest autoencoders - the sort that compress and then reconstruct the original inputs from the compressed representation - we can think of the 'loss' as describing the amount of information lost in the process of reconstruction.

- Illustration:



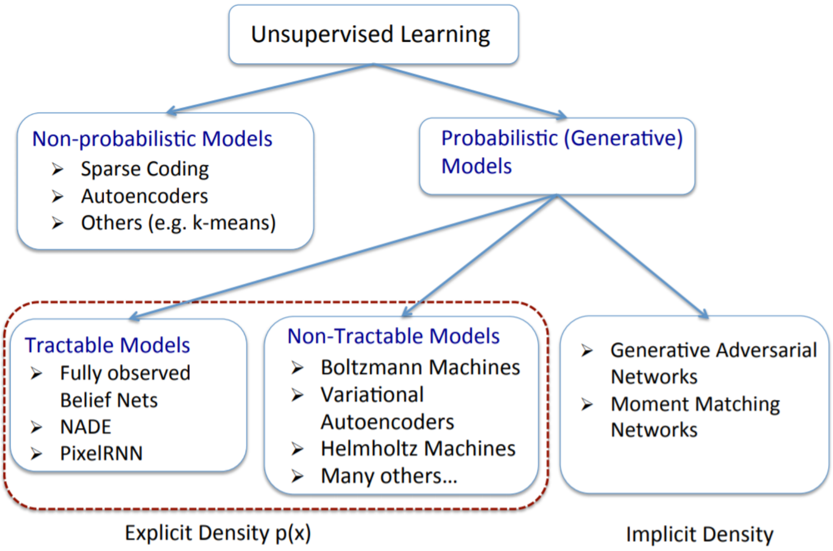
- But what if we want to **generate** data? That is, we assumed that we can represent the high-dimensional observations on a low-dimensional space, can we now find a way to generate these low-dimensional vectors and decode or map them to the high-dimensional space? If we represented images of animals as vectors in $\mathcal{R}^{32}$, can we now create animals just by changing the values of these vectors? This is what **generative models** are trying to achieve.

- Probabilistic methods for describing an observation in latent space allow us to sample those vectors and decode them. Thus, rather than building an encoder which outputs a single value to describe each latent state attribute, the encoder describes a probability distribution for each latent attribute.
- Illustration:

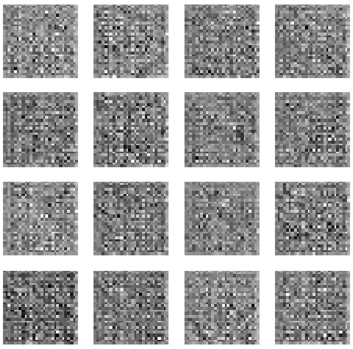


- By constructing the encoder model to output a range of possible values (a statistical distribution) from which can randomly sample to feed into our decoder model, we're essentially enforcing a continuous, smooth latent space representation. For any sampling of the latent distributions, we're expecting our decoder model to be able to accurately reconstruct the input. Thus, values which are nearby to one another in latent space should correspond with very similar reconstructions.

- There are multiple ways to represent and generate data in unsupervised learning as the following diagram shows:



- In unsupervised learning, we separate generative models into two categories:
  - **Explicit generative models** - they model the density of the data, $p(x)$ explicitly. That is, $p(x)$ is a defined distribution, like a Gaussian or any other popular distribution. Variational Autoencoders (VAE) are part of this family (they use Gaussians to model the data).
    -



Source: Convolutional Variational Autoencoder (https://www.tensorflow.org/beta/tutorials/generative/cvae)

- **Implicit generative models** - the data's density is modeled implicitly. That is, $p(x)$ does not have an explicit form and in most modern models, this density is modeled by a deep neural network. Generative Adversary Networks (GAN) are part of this family.
  - ThisPersonDoesNotExist.com (https://thispersondoesnotexist.com) samples from StyleGAN (v2).

Variational Autoencoder (VAE)

As mentioned, there are 2 very popular generative models that are in the core of many research papers and applications:

1. **Generative Adversarial Network (GAN)** - based on concepts from *game theory*, finding the Nash Equilibrium between a discriminator network (D) and a generator network (G). Here, the probability density is modeled *implicitly*.
2. **Variational Autoencoder (VAE)** - based on Bayesian inference, finding (or estimating) the underlying probability distribution of the given data such that it could sample new data from that distribution. Here the probability density is modeled *explicitly*.

In this tutorial we will focus on VAEs (Yay!).

Intuition & Formulation

Let's start with some intuition: we want to generate data (new data, which may not exist like images of never-before seen types of dogs, or cats if you wish). The first step is to decide what type of data you want to generate. For example, let's say we want to generate images of dogs. Then, you imagine how the "new" dog would look like: it must have 4 legs and a tail, nice looking ears and etc... Lastly, you take a sample that matches your cirteria from the distribution of dogs' images.

Let's put a mathematical meaning to all of that:

1. X - the data we want to model (e.g. images of dogs)
2. z - the latent variable (this is the *imagination*, the hidden variable that describes the data, we have seen this before)
3. $P(X)$ - the probability distribution of the data (e.g. the distribution of all dogs' images in the world)
4. $P(z)$ - the probability distribution of the latent variables (the source of the imagination, the brain in this case or the distribution of dogs' images feaures/hidden representations). The **prior**.
5. $P(X|z)$ - the distribution of data generation **given latent variable** (given the features we want the dog to have, the probability of images that satisfy these conditions, turning imagination to real image). The **likelihood**.

The objective is to find $P(X)$ as we wish to build a model for the actual data. Using the **law of total probability**:

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

- Recall that the **joint probability** satisfy: $P(X,z) = P(X|z)P(z)$. Thus, we can say that in the above, we *marginalize* out z from the joint proabibilty distribution $P(X,z)$.
- You have seen similar process in the **EM Algorithm** (or will see later in the course).
- **The problem?** - We don't know $P(X,z)$ or $P(X|z), P(z)$!

Analyzing the ingridients, it makes sense that we need to model $P(z)$ using $P(z|X)$ because we want latent variables that are **likely under the given data** (think of it like this: you want to generate dogs' images, so latent variables of other animals will result in bad generation). This is the idea behind VAEs:

- In VAE we infer $P(z|X)$ using a method calld **Variational Inference (VI)** (hence the name **Variational** Autoencoder).
- **Variational Inference (VI)** - solve an optimization problem in which we model $P(z|X)$ using a simpler distribution, Q, which is easier to evaluate, like a Gaussian, and **minimize the difference between these distributions using the KL-divergence**.

VAE Objective Function - Evidence Lower BOund (ELBO)

- The optimization problem: make the simpler distribution, $Q(z|X)$ as closer as possible to $P(z|X)$.
- The **KL-divergence** is formulated as follows:

$$D_{KL}[Q(z|X)||P(z|X)] = \mathbb{E}_{Q(z|X)}[\log \frac{Q(z|X)}{P(z|X)}]$$

  - $\mathbb{E}_{Q(z|X)}[\log \frac{Q(z|X)}{P(z|X)}] = \sum_z Q(z|X) \log \frac{Q(z|X)}{P(z|X)}$

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

Using **Bayes' Rule**:

$$\rightarrow D_{KL}[Q(z|X)||P(z|X)] = \mathbb{E}_{Q(z|X)}\left[\log Q(z|X) - \log \frac{P(X|z)P(z)}{P(X)}\right]$$

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

- Notice that the expectation is over z and $P(X)$ does not depend on z:

$$\rightarrow D_{KL}[Q(z|X)||P(z|X)] = \mathbb{E}_{Q(z|X)}[\log Q(z|X) - \log P(X|z) - \log P(z)] + \log P(X)$$

Exercise - VAE Objective Function using ELBO

Show that the ELBO of $\log P(X)$ is $\mathbb{E}_{Q(z|X)}[\log P(X|z)] - D_{KL}[Q(z|X)||P(Z)]$, that is, show:

$$\log P(X) \geq \mathbb{E}_{Q(z|X)}[\log P(X|z)] - D_{KL}[Q(z|X)||P(Z)]$$

- Reminder: $D_{KL}[\cdot||\cdot] \geq 0$

Solution

Let's continue from what we found earlier:

$$D_{KL}[Q(z|X)||P(z|X)] = \mathbb{E}_{Q(z|X)}[\log Q(z|X) - \log P(X|z) - \log P(z))] + \log P(X)$$

$$\rightarrow D_{KL}[Q(z|X)||P(z|X)] - \log P(X) = \mathbb{E}_{Q(z|X)}[\log Q(z|X) - \log P(X|z) - \log P(z)]$$

- Move the right-hand side to left and vice versa (or multiply both sides by -1)
$$\log P(X) - D_{KL}[Q(z|X)||P(z|X)] = \mathbb{E}_{Q(z|X)}[\log P(X|z) - (\log Q(z|X) - \log P(z))]$$

$$\log P(X) - D_{KL}[Q(z|X)||P(z|X)] = \mathbb{E}_{Q(z|X)}[\log P(X|z)] - \mathbb{E}_{Q(z|X)}[(\log Q(z|X) - \log P(z))]$$

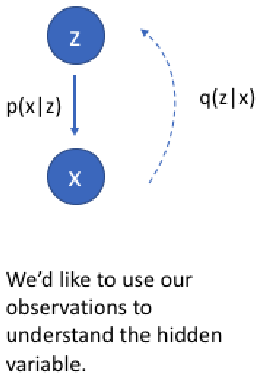
  - Notice: $\mathbb{E}_{Q(z|X)}[(\log Q(z|X) - \log P(z))] = \mathbb{E}_{Q(z|X)}[\log \frac{Q(z|X)}{P(z)}] = D_{KL}[Q(z|X)||P(z)]$

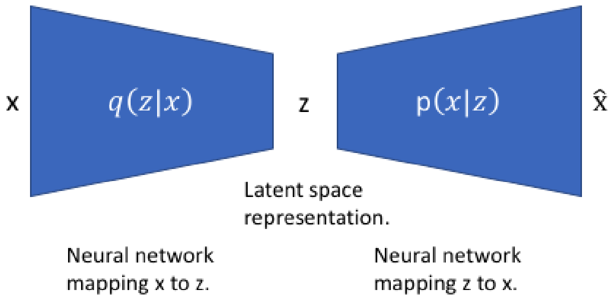
$$\rightarrow \log P(X) - D_{KL}[Q(z|X)||P(z|X)] = \mathbb{E}_{Q(z|X)}[\log P(X|z)] - D_{KL}[Q(z|X)||P(z)]$$

- Since $D_{KL}[Q(z|X)||P(z|X)] \geq 0$ we get:
$$\log P(X) \geq \mathbb{E}_{Q(z|X)}[\log P(X|z)] - D_{KL}[Q(z|X)||P(z)] = ELBO$$

The objective of the VAE is to **maximize the ELBO**, which is in practice good enough as trying to find the exact distribution which is often intractable.

- Notice that since $D_{KL}[Q(z|X)||P(z|X)] \geq 0$, we can say that this is the **error** of the estimation when maximizing the ELBO.
- So why is it called **autoencoder**? Let's put a menaing to each term:
  - $Q(z|X)$ - the **encoder** - given the data X, we want to find the most likely latent space representaion z which is usually in a much lower dimension the the actual data.
  - $P(X|z)$ - the **decoder** - given the latent variable z, we want to generate data that is from the actual distribution.


z
p(x|z)
q(z|x)
x
We'd like to use our observations to understand the hidden variable.


x
$q(z|x)$
z
p$(x|z)$
x̂
Latent space representation.
Neural network mapping x to z.
Neural network mapping z to x.

The *loss* function (which we will soon implement) - if the VAE objective is to **maximize the** $ELBO$ then the loss function would be **minimize the** $(-ELBO)$:
$$\mathcal{L}_{VAE} = -\mathbb{E}_{Q(z|X)}[\log P(X|z)] + D_{KL}[Q(z|X)||P(z)]$$

Analyzing The Objective Components

Let's break the objective function apart and understand what each part means:

- **Reconstruction Loss**: $\mathbb{E}_{Q(z|X)}[\log P(X|z)]$ - this is also called the **log-likelihood** of X under z. Maximizing the likelihood is a well-known concept from Machine Learning course, as **Maximum Likelihood Estimation (MLE)**. You have seen this many times in *supervised* learning settings like *Linear Regression* or *Logistic Regression*.

- Reminder from ML course: the connection between Linear Regression and MLE -
  - Maximum Likelihood Estimation (MLE) is the most common way to estimate parameters of a statistical model by calculating:
$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \log p(y|x, \theta)$$
  - The Negative Log Likelihood (NLL) under **i.i.d** assumption:
$$NLL(\theta) = -\log p(D|\theta) = -\sum_{i=1}^{n} \log p(y_i|x_i, \theta)$$

- If we assume that
$$P(y|x,\theta) = \mathcal{N}(\theta^T x, \sigma^2)$$
  - Note: Why would we assume that? When collecting data from the real world, it is safe to assume that it would not be deterministic, that is, we assume that there is some underlying distribution. More formally, in the case of linear regression, we assume that the y is indeed a linear deterministic function of the input, but with added noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ which leads to $y = \theta^T x + \epsilon \rightarrow p(y|x,\theta) = \mathcal{N}(\theta^T x, \sigma^2)$
  - We get:
$$NLL(\theta) = -\sum_{i=1}^{n} \log\left[(\frac{1}{2\pi\sigma^2})^{\frac{1}{2}} e^{-\frac{1}{2\sigma^2}(y_i - \theta^T x_i)^2}\right] = -\frac{N}{2}\log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \theta^T x_i)^2$$

- Thus, **maximizing the log-likelihood** is equivalent to **minimizing the negative-log-likelihood** w.r.t to θ:
$$\underset{\theta}{\text{argmin}}\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \theta^T x_i)^2 = \underset{\theta}{\text{argmin}}\sum_{i=1}^{n}(y_i - \theta^T x_i)^2 = \underset{\theta}{\text{argmin}}\frac{1}{n}\sum_{i=1}^{n}(y_i - \theta^T x_i)^2 = MSE(\theta)$$
  - The equalities are since σ, n do not depend on θ

- So $\log P(X|z)$ is just the **reconstruction loss** when assuming the data comes from a noisy normal distribution, and in this case it is just the **MSE**: $(\phi(x,z) - x)^2$, where $\phi(x,z)$ is the output of the **decoder**.
  - We can do this with any kind of distribution to get different loss functions!
    - For continuous data, it is better to use MSE (noisy normal distribution) or L_1 loss (noisy Laplace distribution).
    - For discrete/images data, it is better to use **Binary Cross Entropy (BCE)** loss (the data comes from a *Bernoulli* distribution).

- **KL-divergence Loss**: $D_{KL}[Q(z|X)||P(z)]$ - recall that $P(z)$ is the latent variable distribution (the *prior*). We want our VAE to be able to sample random z's, so we want it to be as simple as possible. We model (usually) $z \sim \mathcal{N}(0,1)$ and by minimzing this value we encourage $Q(z|X)$ to be as close as possible to $\mathcal{N}(0,1)$.
  - Another benefit of modeling $P(z)$ as $\mathcal{N}(0,1)$ is that by modeling $Q(z|X)$ as a **Gaussian** with parameters $\mu(X), \Sigma(X)$, that is, the mean and variance given X, then the KL-divergence has a **closed-form soultion**!

Exercise - VAE Objective Closed-Form Solution

Having:

- $z \sim \mathcal{N}(0,1) = P(z)$
- $z|X \sim \mathcal{N}(\mu(X), \Sigma(X)) = Q(z|X)$

Derive the closed-form solution of $D_{KL}[Q(z|X)||P(z)]$.

Solution

We are making the following assumptions:

- The latent space variables are **independent**.
- As a result, $\Sigma(X)$ is a **diagonal** matrix, thus, this matrix can be represented as a **vector**, where the entries are the diagonal of the original matrix.
- Reminders:
  - The i^{th} variable variance, which also the ii entry in the covariance matrix $\Sigma(X)$ satisfies:
$$\Sigma(X)_{ii} = \mathbb{E}[z_i^2] - \mu(X)_i^2 \rightarrow \mathbb{E}[z_i^2] = \Sigma(X)_{ii} + \mu(X)_i^2$$
  - The determinant of a diagonal matrix:
$$|\Sigma(X)| = \prod_{i=1}^{d}\Sigma(X)_{ii}$$

We will solve for the general case $z \in \mathcal{R}^d$ and remember that we represent $\Sigma(X)$ as a vector. We denote $\Sigma = \Sigma(X)$.

$$D_{KL}[Q(z|X)||P(z)] = D_{KL}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I_d)]$$

$$= \mathbb{E}_{z|X}\left[\log \frac{(2\pi)^{-\frac{d}{2}}|\Sigma|^{-\frac{1}{2}} \cdot e^{-\frac{1}{2}(z-\mu(X))^T\Sigma^{-1}(z-\mu(X))}}{(2\pi)^{-\frac{d}{2}}e^{-\frac{1}{2}z^Tz}}\right]$$

$$= \mathbb{E}_{z|X}[-\frac{1}{2}\log|\Sigma| - \frac{1}{2}(z-\mu(X))^T\Sigma^{-1}(z-\mu(X)) + \frac{1}{2}z^Tz]$$

$$= -\frac{1}{2}\log|\Sigma| - \frac{1}{2}\mathbb{E}_{z|X}[z^T\Sigma^{-1}z - 2z^T\Sigma^{-1}\mu(X) + \mu(X)^T\Sigma^{-1}\mu(X) - z^Tz]$$

$$= -\frac{1}{2}\log|\Sigma| - \frac{1}{2}\mu(X)^T\Sigma^{-1}\mu(X) + (\mathbb{E}_{z|X}[z])^T\Sigma^{-1}\mu(X) - \frac{1}{2}\mathbb{E}_{z|X}[z^T\Sigma^{-1}z - z^Tz]$$

$$= -\frac{1}{2}\sum_{i=1}^{d}\log\Sigma(X)_{ii} + \frac{1}{2}\sum_{i=1}^{d}\frac{\mu(X)_i^2}{\Sigma(X)_{ii}} - \frac{1}{2}\mathbb{E}_{z|X}[\sum_{i=1}^{d}\frac{z_i^2}{\Sigma(X)_{ii}} - \sum_{i=1}^{d}z_i^2]$$

$$= -\frac{1}{2}\sum_{i=1}^{d}\log\Sigma(X)_{ii} + \frac{1}{2}\sum_{i=1}^{d}\frac{\mu(X)_i^2}{\Sigma(X)_{ii}} - \frac{1}{2}\sum_{i=1}^{d}[1 + \frac{\mu(X)_i^2}{\Sigma(X)_{ii}} - \mu(X)_i^2 - \Sigma(X)_{ii}]$$

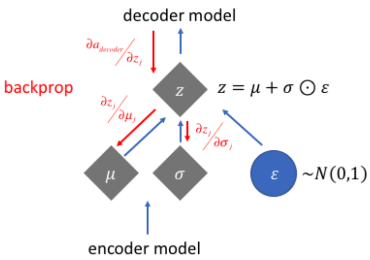
$$= \frac{1}{2}\sum_{i=1}^{d}[\Sigma(X)_{ii} + \mu(X)_i^2 - 1 - \log\Sigma(X)_{ii}]$$

- Note: for **numerical stability**, the netowork outputs $\log\Sigma(X)$ instead of $\Sigma(X)$, and then in the loss function we will take the exponential which is more stable than taking the log.

The Reparameterization Trick

As you recall, in deep neural networks we use **backpropagation** of the gradients to update the weights. In the training process we need to **sample** z's and forward them through the decoder, and they are sampled from $\mathcal{N}(\mu(X), \Sigma(X))$. So normaly, code-wise it would look something like this: `z = torch.normal(mu_x, sigma_x)` or `z = np.normal(mu_x, sigma_x)` .

- What is the problem with that operation?
  - The sampling operation **does not have a gradient!** So we cannot update the encoder with respect to the loss function!
- Solution - **The Reparametrization Trick**:
  - It makes the network differentiable!
  - The trick is as follows:
    - Recall that if you have $x \sim \mathcal{N}(\mu, \Sigma)$ and then you perform standartization, x_{std}, so that $\mu = 0, \Sigma = 1$, then you can revert it back to the original distribution by: $x = \mu + \Sigma^{\frac{1}{2}}x_{std}$.
    - In our case, let $\epsilon \sim \mathcal{N}(0, 1)$:
    $$z = \mu(X) + \Sigma(X)^{\frac{1}{2}}\epsilon$$
    - No we can take the derivative w.r.t. to $\mu(X), \Sigma(X)$ and backpropagate it through the network!


decoder model
$\partial a_{decoder}/\partial z_j$
backprop
z
$z = \mu + \sigma \odot \varepsilon$
$\partial z_j/\partial \mu_j$
$\partial z_j/\partial \sigma_j$
μ
σ
ε
$\sim N(0,1)$
encoder model

Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- Variational Inference (VI) - Machine Learning: Variational Inference (https://www.youtube.com/watch?v=2pEkWk-LHmU)
  - Until 13:30 mins
- Analyzing the KL-Divergence in VI - Variational Inference Part 2 (KL divergence) (https://www.youtube.com/watch?v=uKxtmkfeuxg)
- Generative Models (VAEs + GANs) - Standord CS231n - Lecture 13 I Generative Models (https://www.youtube.com/watch?v=5WoItGTWV54)

Credits

- Deep Learning - Unsupervised Learning, Tutorial by Ruslan Salakhutdinov (CMU) (https://www.cs.cmu.edu/~rsalakhu/talk_MLSS_part2.pdf) - https://www.cs.cmu.edu/~rsalakhu/ (https://www.cs.cmu.edu/~rsalakhu/)
- CS294-158 Deep Unsupervised Learning Spring 2019 (https://sites.google.com/view/berkeley-cs294-158-sp19/home) @ UC Berkeley - https://sites.google.com/view/berkeley-cs294-158-sp19/home (https://sites.google.com/view/berkeley-cs294-158-sp19/home)
- Variational autoencoders. (https://jeremyjordan.me/variational-autoencoders/) by Jeremy Jordan - https://jeremyjordan.me (https://jeremyjordan.me)
- Variational Autoencoder: Intuition and Implementation (https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/) by Agustinus Kristiadi
- Icons from Icon8.com (https://icons8.com/) - https://icons8.com (https://icons8.com)
- Datasets from Kaggle (https://www.kaggle.com/) - https://www.kaggle.com/ (https://www.kaggle.com/)