Naive Bayes Classifiers
Removing stop words with NLTK in Python
Linear Regression (Python Implementation)
Agents in Artificial Intelligence
Decision Tree
Supervised and Unsupervised learning
Decision Tree Introduction with example
Reinforcement learning
ML | Linear Regression
Search Algorithms in AI
Python | Decision tree implementation
Elbow Method for optimal value of k in KMeans
Convert Text to Speech in Python
Activation functions in Neural Networks
Getting started with Machine Learning

# Role of KL-divergence in Variational Autoencoders

Last Updated : 27 Jan, 2022

## Variational Autoencoders

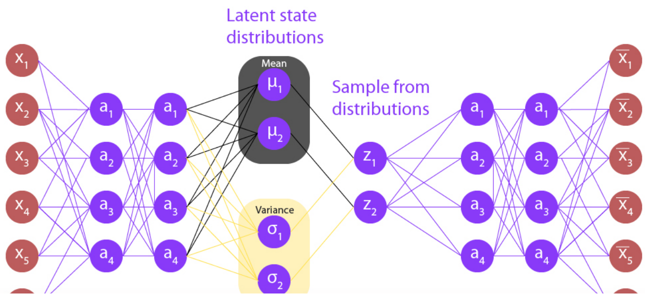
Variational autoencoder was proposed in 2013 by Knigma and Welling at Google and Qualcomm. A variational autoencoder (VAE) provides a probabilistic manner for describing an observation in latent space. Thus, rather than building an encoder that outputs a single value to describe each latent state attribute, we'll formulate our encoder to describe a probability distribution for each latent attribute.

**Architecture:**

Autoencoders basically contains two parts:

- The first one is an encoder which is similar to the convolution neural network except for the last layer. The aim of the encoder is to learn efficient data encoding from the dataset and pass it into a bottleneck architecture.
- The other part of the autoencoder is a decoder that uses latent space in the bottleneck layer to regenerate the images similar to the dataset. These results backpropagate from the neural network in the form of the loss function.

Variational autoencoder is different from autoencoder in a way such that it provides a statistical manner for describing the samples of the dataset in latent space. Therefore, in the variational autoencoder, the encoder outputs a probability distribution in the bottleneck layer instead of a single output value.




Courses Tutorials Jobs Practice Contests Sign In
Read Discuss

**Explanation:**

The Variational Autoencoder latent space is continuous. It provides random sampling and interpolation. Instead of outputting a vector of size n, the encoder outputs two vectors:

Vector ???? of means (vector size n)
Vector ???? of standard deviations (vector size n)

Output is an approximate posterior distribution q(z|x). Sample from this distribution to get z. let's look at more details into Sampling:

Let's take some values of mean and standard deviation,

$$\mu = [0.15, 1.1, 0.2, 0.7, ...]$$
$$\sigma = [0.2, 0.4, 0.5, 2.4, ...]$$

The intermediate distribution that is generated from that:

$$X = [X_1 \sim \mathbf{N}(0.15, 0.2^2), X_2 \sim \mathbf{N}(1.1, 0, 4^2), X_3 \sim \mathbf{N}(0.2, 0, 5^2), X_4 \sim \mathbf{N}(0.7, 1.5^2), ...]$$

Now, let's generate the sampled vector from this:

$$[0.32, 1.25, 0.8, 2.0]$$

While the mean and standard deviation are the same for one input the result may be different due to sampling. Eventually, our goal is to make the encoder learn to generate differently???? For different classes, clustering them and generating encoding such they don't vary much. To this we use KL-divergence.

**KL-divergence:**

KL divergence stands for Kullback Leibler Divergence, it is a measure of divergence between two distributions. Our goal is to Minimize KL divergence and optimize ???? and ???? of one distribution to resemble the required distribution.Of

For multiple distribution the KL-divergence can be calculated as the following formula:

$$\frac{1}{2}\sum_{j=1}^{J}(1 + log(\sigma_j^{(i)}))\frac{1}{2}\sum_{j=1}^{J}(1 + log(\sigma_j^{(i)})^2) - (\mu_j^{(i)})^2 - (\sigma_j^{(i)})^2)$$

where X_j \sim N(\mu_j, \sigma_j^{2}) is the standard normal distribution.

**VAE Loss:**

Suppose we have a distribution z and we want to generate the observation x from it.  In other words, we want to calculate

$p(z|x)$

We can do it by following way:

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

But, the calculation of p(x) can be quite difficult

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

This usually makes it an intractable distribution. Hence, we need to approximate p(z|x) to q(z|x) to make it a tractable distribution. To better approximate p(z|x) to q(z|x), we will minimize the KL-divergence loss which calculates how similar two distributions are:

$\min KL(q(z|x)||p(z|x))$

By simplifying, the above minimization problem is equivalent to the following maximization problem :

$E_{q(z|x)} \log p(x|z) - KL(q(z|x)||p(z))$

The first term represents the reconstruction likelihood and the other term ensures that our learned distribution q is similar to the true prior distribution p.

Thus our total loss consists of two terms, one is reconstruction error and the other is KL-divergence loss:

$Loss = L(x, \hat{x}) + \sum_j KL(q_j(z|x)||p(z))$

**Implementation:**

In this implementation, we will be using the MNIST dataset, this dataset is already available in *keras.datasets* API, so we don't need to add or upload manually.

First, we need to import the necessary packages to our python environment. we will be using Keras package with TensorFlow as a backend.

## python3

```
# code
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import Input, Model
from tensorflow.keras.layers import Layer, Conv2D
import matplotlib.pyplot as plt
```

For variational autoencoders, we need to define the architecture of two parts encoder and decoder but first, we will define the bottleneck layer of architecture, the sampling layer.

## python3

```
# this sampling layer is the bottleneck layer of
# it uses the output from two dense layers z_mean
# convert them into normal distribution and pass
class Sampling(Layer):

    def call(self, inputs):
        z_mean, z_log_var = inputs
        batch = tf.shape(z_mean)[0]
        dim = tf.shape(z_mean)[1]
        epsilon = tf.keras.backend.random_normal(
        return z_mean + tf.exp(0.5 * z_log_var) *
```

Now, we define the architecture of the encoder part of our autoencoder, this part takes images as input and encodes their representation in the Sampling layer.

## python3

```
# Define Encoder Model
latent_dim = 2

encoder_inputs = Input(shape =(28, 28, 1))
x = Conv2D(32, 3, activation ="relu", strides = 2
x = Conv2D(64, 3, activation ="relu", strides = 2
x = Flatten()(x)
x = Dense(16, activation ="relu")(x)
z_mean = Dense(latent_dim, name ="z_mean")(x)
z_log_var = Dense(latent_dim, name ="z_log_var")(
z = Sampling()([z_mean, z_log_var])
encoder = Model(encoder_inputs, [z_mean, z_log_va
encoder.summary()
```

```
Model: "encoder"
________________________________________________
```

```
Layer (type)                 Output Shape
==================================================
input_3 (InputLayer)         [(None, 28, 28, 1
__________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32
__________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 64)
__________________________________________________
flatten_1 (Flatten)          (None, 3136)
__________________________________________________
dense_2 (Dense)              (None, 16)
__________________________________________________
z_mean (Dense)               (None, 2)
__________________________________________________
z_log_var (Dense)            (None, 2)
__________________________________________________
sampling_1 (Sampling)        (None, 2)

==================================================
Total params: 69, 076
Trainable params: 69, 076
Non-trainable params: 0
__________________________________________________
```

Now, we define the architecture of decoder part of our autoencoder, this part takes the output of the sampling layer as input and output an image of size (28, 28, 1) .

**python3**

```
# Define Decoder Architecture
latent_inputs = keras.Input(shape =(latent_dim, )
x = Dense(7 * 7 * 64, activation ="relu")(latent_
x = Reshape((7, 7, 64))(x)
x = Conv2DTranspose(64, 3, activation ="relu", st
x = Conv2DTranspose(32, 3, activation ="relu", st
decoder_outputs = Conv2DTranspose(1, 3, activatio
decoder = Model(latent_inputs, decoder_outputs, n
decoder.summary()
```

```
Model: "decoder"
__________________________________________________
Layer (type)                 Output Shape
==================================================
input_4 (InputLayer)         [(None, 2)]
__________________________________________________
dense_3 (Dense)              (None, 3136)
__________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 64)
__________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 14, 14, 64)
__________________________________________________
conv2d_transpose_4 (Conv2DTr (None, 28, 28, 32)
```

```
_________________________________________________________________
conv2d_transpose_5 (Conv2DTr (None, 28, 28, 1)
=================================================================
Total params: 65, 089
Trainable params: 65, 089
Non-trainable params: 0
_________________________________________________________________
```

In this step, we combine the model and define the training procedure with loss functions.

## python3

```
# this class takes encoder and decoder models and
# define the complete variational autoencoder arc
class VAE(keras.Model):
    def __init__(self, encoder, decoder, **kwargs
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    def train_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = encoder(data)
            reconstruction = decoder(z)
            reconstruction_loss = tf.reduce_mean(
                keras.losses.binary_crossentropy(
            )
            reconstruction_loss *= 28 * 28
            kl_loss = 1 + z_log_var - tf.square(z
            kl_loss = tf.reduce_mean(kl_loss)
            kl_loss *= -0.5
            # beta =10
            total_loss = reconstruction_loss + 10
        grads = tape.gradient(total_loss, self.tr
        self.optimizer.apply_gradients(zip(grads,
        return {
            "loss": total_loss,
            "reconstruction_loss": reconstruction
            "kl_loss": kl_loss,
        }
```

Now it's the right time to train our variational autoencoder model, we will train it for 100 epochs.  But first we need to import the MNIST dataset.

## python3

```
# load fashion mnist dataset  from  keras.dataset
(x_train, _), (x_test, _) = keras.datasets.fashio
fmnist_images = np.concatenate([x_train, x_test],
# expand dimension to add  a color map dimension
fmnist_images = np.expand_dims(fmnist_images, -1)

# compile and train the model
vae = VAE(encoder, decoder)
vae.compile(optimizer ='rmsprop')
vae.fit(fmnist_images, epochs = 100, batch_size =
```

In this step, we display training results, we will be displaying these results according to their values in latent space vectors.

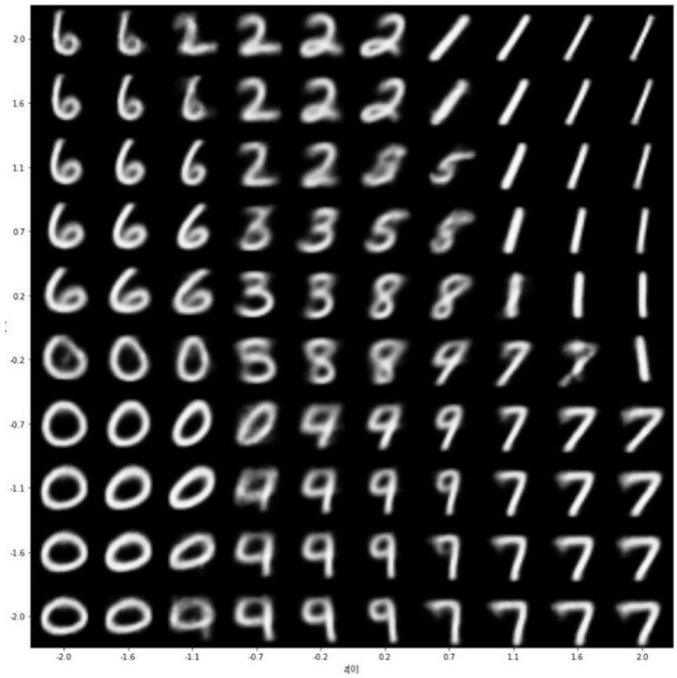
## python3

```
def plot_latent(encoder, decoder):
    # display a n * n 2D manifold of images
    n = 10
    img_dim = 28
    scale = 2.0
    figsize = 15
    figure = np.zeros((img_dim * n, img_dim * n))
    # linearly spaced coordinates corresponding t
    # of images classes in the latent space
    grid_x = np.linspace(-scale, scale, n)
    grid_y = np.linspace(-scale, scale, n)[::-1]

    for i, yi in enumerate(grid_y):
        for j, xi in enumerate(grid_x):
            z_sample = np.array([[xi, yi]])
            x_decoded = decoder.predict(z_sample)
            images = x_decoded[0].reshape(img_dim
            figure[
                i * img_dim : (i + 1) * img_dim,
                j * img_dim : (j + 1) * img_dim,
            ] = images

    plt.figure(figsize =(figsize, figsize))
    start_range = img_dim // 2
    end_range = n * img_dim + start_range + 1
    pixel_range = np.arange(start_range, end_rang
    sample_range_x = np.round(grid_x, 1)
    sample_range_y = np.round(grid_y, 1)
    plt.xticks(pixel_range, sample_range_x)
    plt.yticks(pixel_range, sample_range_y)
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.imshow(figure, cmap ="Greys_r")
    plt.show()


plot_latent(encoder, decoder)
```



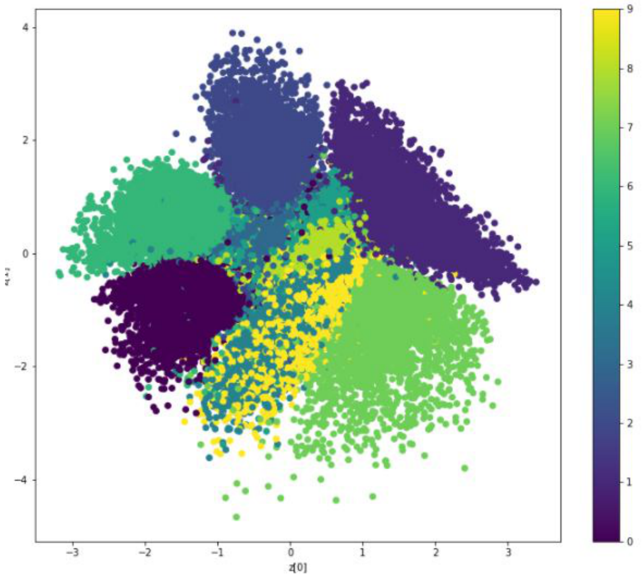
*Output from Encoder*

To get a more clear view of our representational latent vectors values, we will be plotting the scatter plot of training data on the basis of their values of corresponding latent dimensions generated from the encoder

## python3

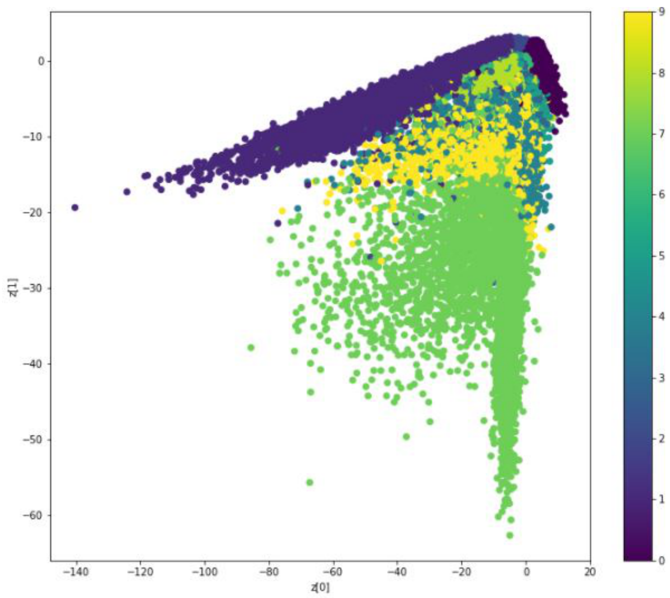
```
def plot_label_clusters(encoder, decoder, data, t
    z_mean, _, _ = encoder.predict(data)
    plt.figure(figsize =(12, 10))
    sc = plt.scatter(z_mean[:, 0], z_mean[:, 1],
    cbar = plt.colorbar(sc, ticks = range(10))
    cbar.ax.set_yticklabels([i for i in range(10)
    plt.xlabel("z[0]")
    plt.ylabel("z[1]")
    plt.show()

(x_train, y_train), _ = keras.datasets.mnist.load
x_train = np.expand_dims(x_train, -1).astype("flo
plot_label_clusters(encoder, decoder, x_train, y_
```




*Distribution for Beta = 10*

To compare the difference, I also train the above autoencoder for \beta =0 i.e we remove the Kl-divergence loss, and it generated the following distribution:




*Distribution for Beta = 0*

Here, we can see that the distribution is not separable and quite skewed for different values, that's why we use KL-divergence loss in the above variational autoencoder.

**References:**

[Variational Autoencoder Paper]
[Keras Variational Autoencoder]

Like 2

Previous

Python - Gaussian fit

Next

Face and Hand Landmarks Detection using Python - Mediapipe, OpenCV

RECOMMENDED ARTICLES Page : 1 2 3

01 Variational AutoEncoders
20, Jul 20

02 Disentanglement in Beta Variational Autoencoders
18, Sep 21

03 ML | Variational Bayesian Inference for Gaussian Mixture
12, Jul 19

04 How Autoencoders works ?
15, May 19

05 Colorization Autoencoders using Keras
07, Jun 20

06 PyQt5 - Getting the role of selected item in ComboBox
02, Apr 20

07 What is the Role of Planning in Artificial Intelligence?
04, Nov 19

08 Machine Learning Role in Business Growth and Development
12, Feb 20

Article Contributed By :

pawangfg
@pawangfg

Vote for difficulty

Easy Normal Medium Hard Expert

Improved By : simmytarika5

Article Tags : Neural Network, Machine Learning, Python

Practice Tags : Machine Learning, python

Improve Article Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

**Company**
About Us
Careers
In Media
Contact Us
Privacy Policy
Copyright Policy

**Learn**
Algorithms
Data Structures
SDE Cheat Sheet
Machine learning
CS Subjects
Video Tutorials
Courses

**News**
Top News
Technology
Work & Career
Business
Finance
Lifestyle
Knowledge

**Languages**
Python
Java
CPP
Golang
C#
SQL
Kotlin

**Web Development**
Web Tutorials
Django Tutorial
HTML
JavaScript
Bootstrap
ReactJS
NodeJS

**Contribute**
Write an Article
Improve an Article
Pick Topics to Write
Write Interview Experience
Internships
Video Internship