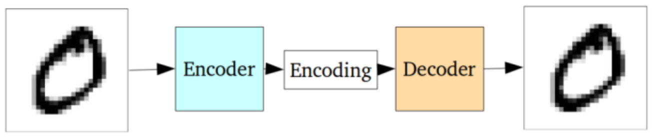
Variational Autoencoders Learning Note

Ruoqi Wei

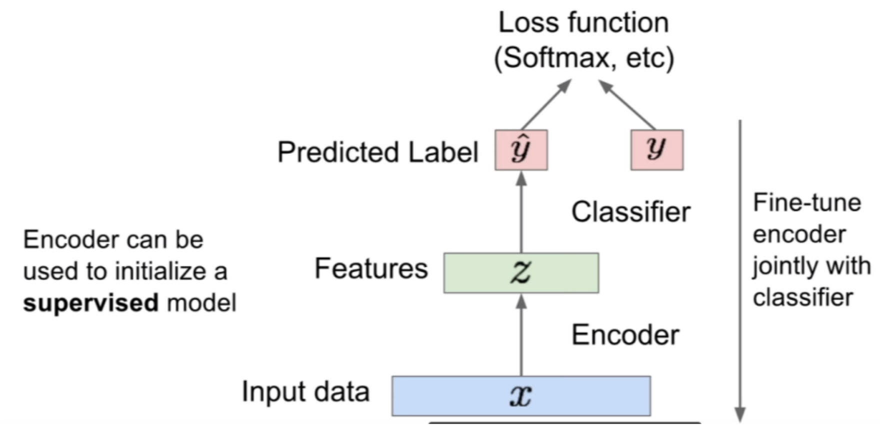
Part 1.Intuitively Understanding

- Model definition:
  Probability distribution Generative model that generate a random, new output, that looks similar to the training data.（reconstruct original data）
- Components:
  An autoencoder network is a pair of two connected networks, an encoder and a decoder. An encoder network takes in an input, and converts it into a smaller features, which the decoder network can use to convert it back to the original input.



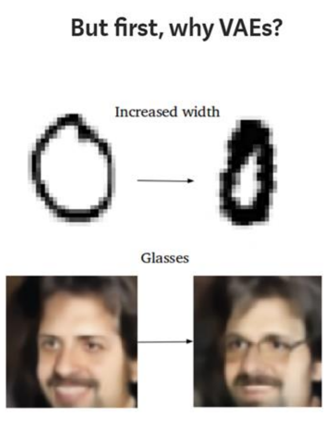
A standard Autoencoder

- The use of the Model :
  Dimensionality reduction, increased Width, Missing partial filling for better recognition .etc.
- The use of the Model in Computer Vision:
  In computer vision,it can be used to initialize a supervised model with better features as a Fine-tune encoder jointly with classifier.

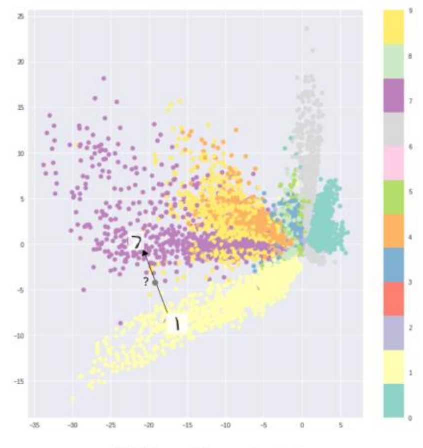

Loss function
(Softmax, etc)
Predicted Label
$\hat{y}$
y
Classifier
Fine-tune encoder jointly with classifier
Encoder can be used to initialize a **supervised** model
Features
z
Encoder
Input data
x

- Why VAEs?

When training a generative model, the more complicated the dependencies between the dimensions, the more difficult the models are to train. The autoencoder takes in an input and produces a much smaller representation that contains enough information for the next part of the network, so that making the next part of the model easier to train.



Exploring a specific variation of input data[1]



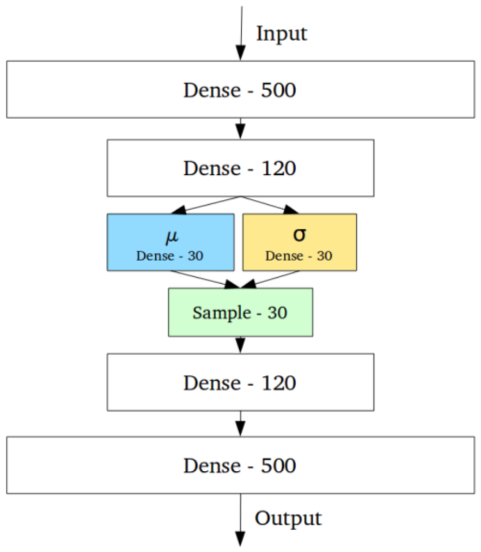
Optimizing purely for reconstruction loss

- Compare to standard autoencoder,what's the advantages of VAEs?

The problem with standard autoencoders is that the latent space may not be continuous, or allow easy interpolation,which will simply generate an unrealistic output. The VAEs Improved this shortage by making the latent spaces continuous and allowing easy random sampling and interpolation.

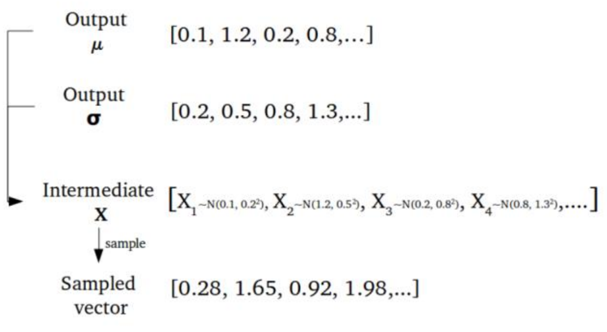
- How VAEs achieved?

1. First,it achieves this by outputting two vectors of size n: a vector of means, μ, and another vector of standard deviations, σ.



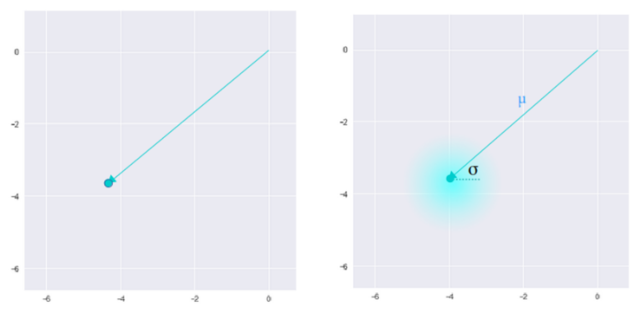
Variational Autoencoder

2. They form the parameters of a vector of random variables of length n, with the i th element of μ and σ being the mean and standard deviation of the i_{th} random variable, X_i, from which we sample, to obtain the sampled encoding which we pass onward to the decoder:



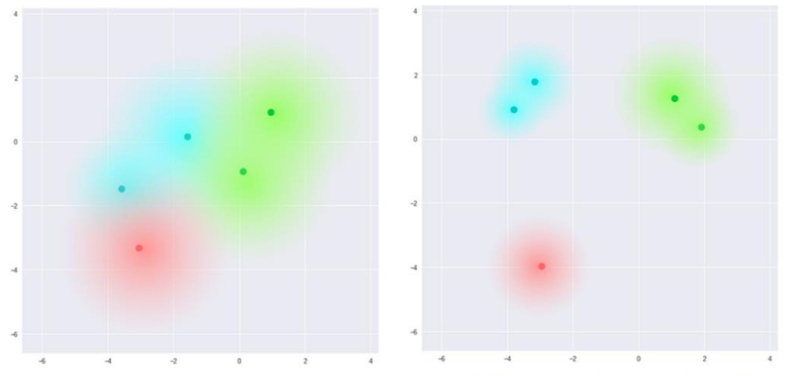
Stochastically generating encoding vectors

3. This stochastic generation means, that even for the same input, while the mean and standard deviations remain the same, the actual encoding will somewhat vary on every single pass simply due to sampling.



Standard Autoencoder (direct encoding coordinates)   Variational Autoencoder (μ and σ initialize a probability distribution)

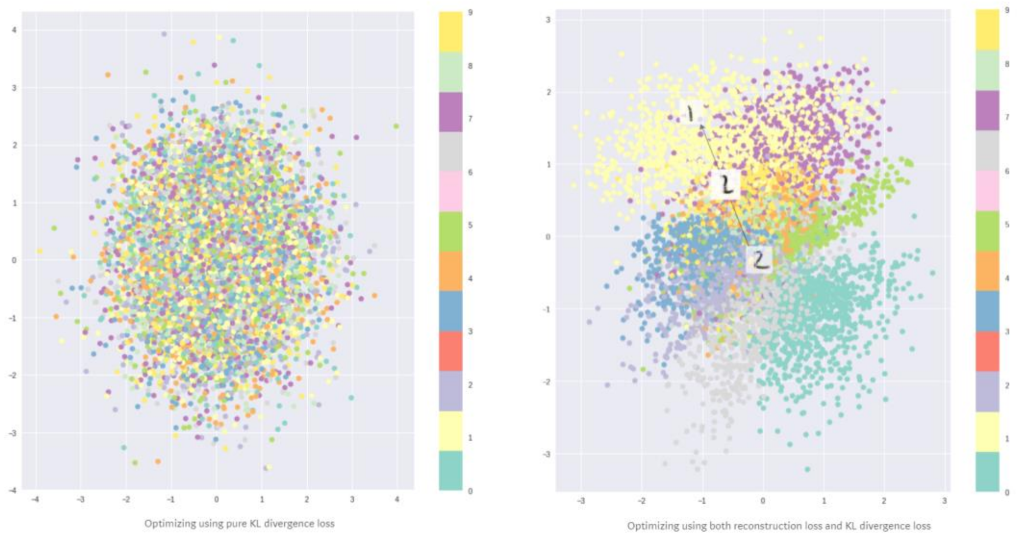
4. The mean vector controls where the encoding of an input should be centered around, while the standard deviation controls the "area".


What we require   What we may inadvertently end up with

5. What we ideally want is that the encodings are as close as possible to each other while still being

different, allowing smooth interpolation and enabling the construction of new samples.In order to force this, we use the Kullback–Leibler divergence (KL divergence) into the loss function.

6. Minimizing the KL divergence means optimizing the probability distribution parameters (μ and σ) to closely resemble that of the target distribution.



7. If just using KL loss itself ,this would create meaningless results,because the decoder can not get meaningful information.
8. Therefore, it is necessary to combine the KL loss with the reconstruction loss. This allows not only the local samples can stay in the same category in the latent space, but also all samples in the global scope are compactly compressed into the continuous latent space.

Part 2. Mathematical Basics

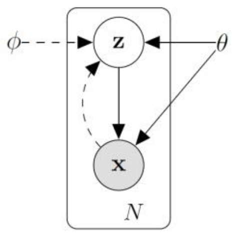


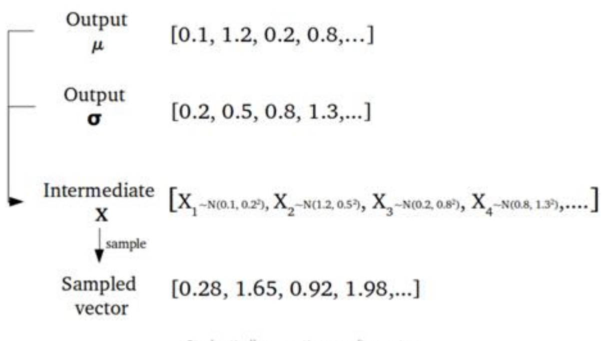
Figure 1: The type of directed graphical model under consideration. Solid lines denote the generative model $p_{\theta}(\mathbf{z})p_{\theta}(\mathbf{x}|\mathbf{z})$, dashed lines denote the variational approximation $q_{\phi}(\mathbf{z}|\mathbf{x})$ to the intractable posterior $p_{\theta}(\mathbf{z}|\mathbf{x})$. The variational parameters ϕ are learned jointly with the generative model parameters θ.

The figure above is a Probabilistic Graphical Model(PGM) of VAEs. The sample we can observe is x, and x is generated by the latent variable z. From z to x is the generative model.

- Statistical Learning

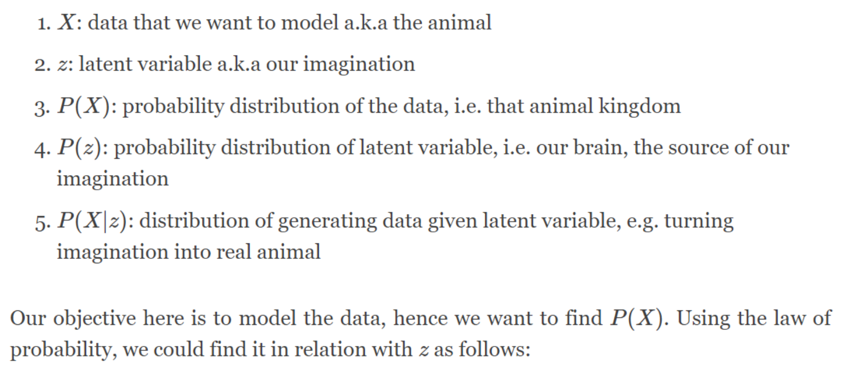
Model: Generative Model (conditional probability distribution).
Strategy: Minimizing the difference between distributions.
Algorithm: Optimizing KL divergence +Maximizing “ELBO”(Lower bound)

- Latent Variable:My understanding z is the random variable sampled vector as shown below.



Stochastically generating encoding vectors

- Next,let’s define some notions:



1. X: data that we want to model a.k.a the animal
2. z: latent variable a.k.a our imagination
3. $P(X)$: probability distribution of the data, i.e. that animal kingdom
4. $P(z)$: probability distribution of latent variable, i.e. our brain, the source of our imagination
5. $P(X|z)$: distribution of generating data given latent variable, e.g. turning imagination into real animal

Our objective here is to model the data, hence we want to find $P(X)$. Using the law of probability, we could find it in relation with z as follows:

- 1. Model the data -Find P(X). (hypothesis)
  Our goal is to model the data, so we want to find **P(X).** Using the law of probability, we could find it in relation with z as follows:

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

  that is, we marginalize out z from the joint probability distribution P(X,z).Now we just need to know P(X,z), or equivalently, P(X|z)and P(z).

- 2. Find P(z)using P(z|X)
  The idea of VAE is to infer P(z)using P(z|X) because we want to make the latent variable likely under the original data.

- 3.Find P(z|X) using Q(z|X)

Variational Inference is one of the method in bayesian inference,The main idea of VI is to pose the inference by approach it as an optimization problem. How? By minimizing the difference between those two distribution using KL divergence metric.

Alright, now let's say we want to infer $P(z|X)$ using $Q(z|X)$. The KL divergence then formulated as follows:

$$\begin{aligned} D_{KL}[Q(z|X)\|P(z|X)] &= \sum_z Q(z|X)\,\log\frac{Q(z|X)}{P(z|X)} \\ &= E\left[\log\frac{Q(z|X)}{P(z|X)}\right] \\ &= E[\log Q(z|X) - \log P(z|X)] \end{aligned}$$

Recall the notations above, there are two things that we haven't use, namely $P(X)$, $P(X|z)$, and $P(z)$. But, with Bayes' rule, we could make it appear in the equation:

$$\begin{aligned} D_{KL}[Q(z|X)\|P(z|X)] &= E\left[\log Q(z|X) - \log\frac{P(X|z)P(z)}{P(X)}\right] \\ &= E[\log Q(z|X) - (\log P(X|z) + \log P(z) - \log P(X))] \\ &= E[\log Q(z|X) - \log P(X|z) - \log P(z) + \log P(X)] \end{aligned}$$

- 4.Find the VAE objective function

Notice that the expectation is over z and $P(X)$ doesn't depend on z, so we could move it outside of the expectation.

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

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

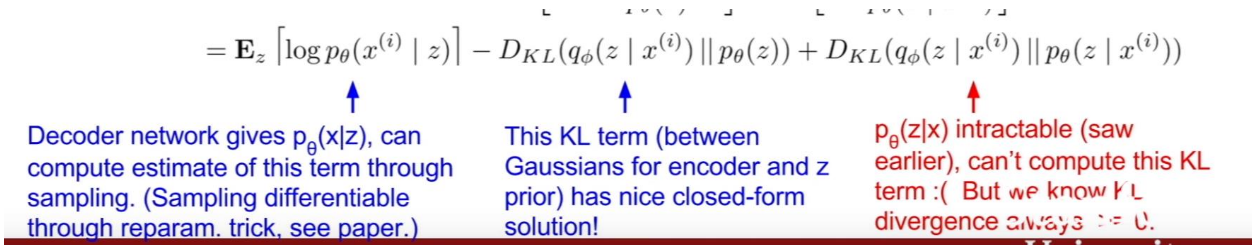
If we look carefully at the right hand side of the equation, we would notice that it could be rewritten as another KL divergence. So let's do that by first rearranging the sign.

$$\begin{aligned} D_{KL}[Q(z|X)\|P(z|X)] - \log P(X) &= E[\log Q(z|X) - \log P(X|z) - \log P(z)] \\ \log P(X) - D_{KL}[Q(z|X)\|P(z|X)] &= E[\log P(X|z) - (\log Q(z|X) - \log P(z))] \\ &= E[\log P(X|z)] - E[\log Q(z|X) - \log P(z)] \\ &= E[\log P(X|z)] - D_{KL}[Q(z|X)\|P(z)] \end{aligned}$$

And this is it, the VAE objective function:

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

- 5. Optimize the objective function
  Optimizing KL divergence metric+Maximize "ELBO"(Lower bound)

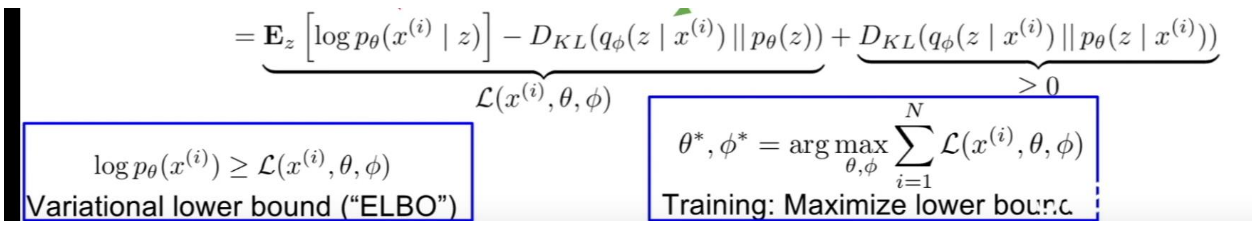


$$= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling. (Sampling differentiable through reparam. trick, see paper.)

This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!

$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always >= 0.



$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + \underbrace{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))}_{\geq 0}$$

$$\boxed{\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)},\theta,\phi)}$$

Variational lower bound ("ELBO")

$$\boxed{\theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)},\theta,\phi)}$$

Training: Maximize lower bound

5.1 Optimizing KL divergence term:

To make approximate posterior distribution close to prior.

If q takes each dimension's independent Gaussian distribution and makes p a standard normal distribution, then the KL divergence term as:

$$D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] = \frac{1}{2}\sum_k \left(\exp(\Sigma(X)) + \mu^2(X) - 1 - \Sigma(X)\right)$$

Prove:

$$\begin{aligned} D_{KL}[N(\mu(X),\Sigma(X))\|N(0,1)] &= \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \log\prod_k \Sigma(X)\right) \\ &= \frac{1}{2}\left(\sum_k \Sigma(X) + \sum_k \mu^2(X) - \sum_k 1 - \sum_k \log\Sigma(X)\right) \\ &= \frac{1}{2}\sum_k \left(\Sigma(X) + \mu^2(X) - 1 - \log\Sigma(X)\right) \end{aligned}$$

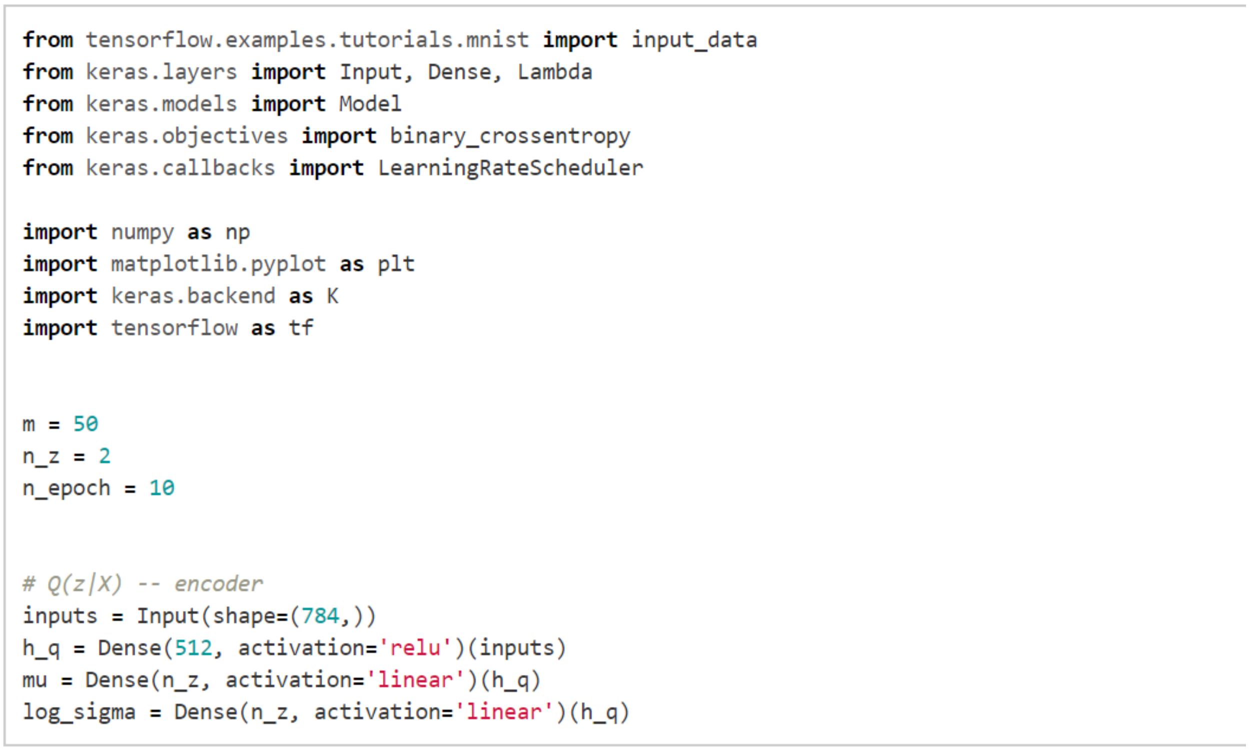
5.2 Maximizing "ELBO"(Lower bound) term:

To reconstruct the input data.

Maximizing the term $E[\log P(X|z)]$ is a maximum likelihood estimation. Decoder network gives p(x|z) can compute estimate of this term through reparameterization trick.(see paper)

Part 3. Implementation

First, let’s implement the encoder net Q(z|X), which takes input X and outputting two things: μ(X) and Σ(X) , the parameters of the Gaussian.



```python
from tensorflow.examples.tutorials.mnist import input_data
from keras.layers import Input, Dense, Lambda
from keras.models import Model
from keras.objectives import binary_crossentropy
from keras.callbacks import LearningRateScheduler

import numpy as np
import matplotlib.pyplot as plt
import keras.backend as K
import tensorflow as tf


m = 50
n_z = 2
n_epoch = 10


# Q(z|X) -- encoder
inputs = Input(shape=(784,))
h_q = Dense(512, activation='relu')(inputs)
mu = Dense(n_z, activation='linear')(h_q)
log_sigma = Dense(n_z, activation='linear')(h_q)
```

That is, our Q(z|X) is a neural net with one hidden layer. In this implementation, our latent variable is two dimensional, so that we could easily visualize it. In practice though, more dimension in latent variable should be better.

However, we are now facing a problem. How do we get z from the encoder outputs? Obviously we could sample z from a Gaussian which parameters are the outputs of the encoder. Alas, sampling directly won’t do, if we want to train VAE with gradient descent as the sampling operation doesn’t have gradient!

There is, however a trick called reparameterization trick, which makes the network differentiable. Reparameterization trick basically divert the non-differentiable operation out of the network, so that, even though we still involve a thing that is non-differentiable, at least it is out of the network, hence the network could still be trained.

The reparameterization trick is as follows. Recall, if we have x~N(μ , Σ)and then standardize it so that μ =0, Σ =1, we could revert it back to the original distribution by reverting the standardization process. Hence, we have this equation:

$$x = \mu + \Sigma^{\frac{1}{2}} x_{std}$$

With that in mind, we could extend it. If we sample from a standard normal distribution, we could convert it to any Gaussian we want if we know the mean and the variance. Hence we could implement our sampling operation of z by:

$$z = \mu(X) + \Sigma^{\frac{1}{2}}(X)\,\epsilon$$

where $\epsilon \sim N(0,1)$.
Now, during backpropagation, we don't care anymore with the sampling process, as it is now outside of the network, i.e. doesn't depend on anything in the net, hence the gradient won't flow through it.

```
def sample_z(args):
    mu, log_sigma = args
    eps = K.random_normal(shape=(m, n_z), mean=0., std=1.)
    return mu + K.exp(log_sigma / 2) * eps


# Sample z ~ Q(z|X)
z = Lambda(sample_z)([mu, log_sigma])
```

Now we create the decoder net P(X|z):

```
# P(X|z) -- decoder
decoder_hidden = Dense(512, activation='relu')
decoder_out = Dense(784, activation='sigmoid')

h_p = decoder_hidden(z)
outputs = decoder_out(h_p)
```

Lastly, from this model, we can do three things: reconstruct inputs, encode inputs into latent variables, and generate data from latent variable. So, we have three Keras models:

```
# Overall VAE model, for reconstruction and training
vae = Model(inputs, outputs)

# Encoder model, to encode input into latent variable
# We use the mean as the output as it is the center point, the representative of the gaussian
encoder = Model(inputs, mu)

# Generator model, generate new data given latent variable z
d_in = Input(shape=(n_z,))
d_h = decoder_hidden(d_in)
d_out = decoder_out(d_h)
decoder = Model(d_in, d_out)
```

Then, we need to translate our loss into Keras code:

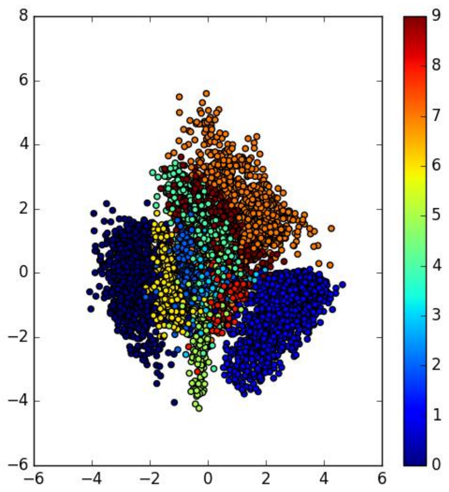
```
def vae_loss(y_true, y_pred):
    """ Calculate loss = reconstruction loss + KL loss for each data in minibatch """
    # E[log P(X|z)]
    recon = K.sum(K.binary_crossentropy(y_pred, y_true), axis=1)
    # D_KL(Q(z|X) || P(z|X)); calculate in closed form as both dist. are Gaussian
    kl = 0.5 * K.sum(K.exp(log_sigma) + K.square(mu) - 1. - log_sigma, axis=1)

    return recon + kl
```

and then train it:

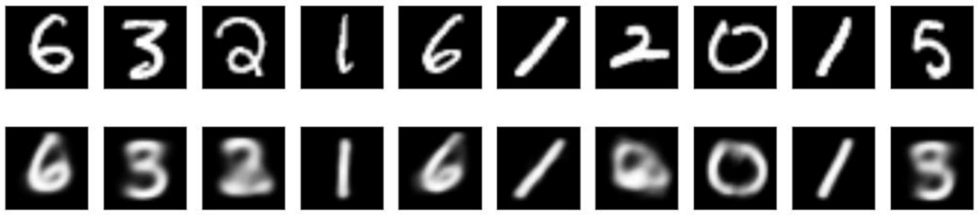
```
vae.compile(optimizer='adam', loss=vae_loss)
vae.fit(X_train, X_train, batch_size=m, nb_epoch=n_epoch)
```

Part.4 Implementation on MNIST Data

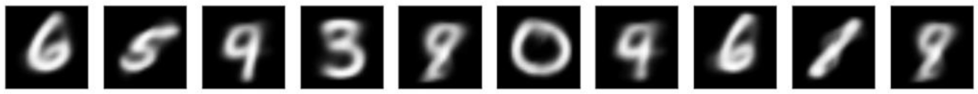
We could use any dataset really, but like always, we will use MNIST as an example.
After we trained our VAE model, we then could visualize the latent variable space Q(z|X):



As we could see, in the latent space, the representation of our data that have the same characteristic, e.g. same label, are close to each other. Notice that in the training phase, we never provide any information regarding the data.
We could also look at the data reconstruction by running through the data into overall VAE net:



Lastly, we could generate new sample by first sample z~N(0,1)and feed it into our decoder net:



If we look closely on the reconstructed and generated data, we would notice that some of the data are ambiguous. For example the digit 5 looks like 3 or 8. That's because our latent variable space is a continous distribution (i.e. N(0,1)), hence there bound to be some smooth transition on the edge of the clusters. And also, the cluster of digits are close to each other if they are somewhat similar. That's why in the latent space, 5 is close to 3.

References:

- https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf
- https://wiseodd.github.io/techblog/2016/12/10/variational-autoencoder/
- https://medium.com/@cotra.marko/making-sense-of-the-kullback-leibler-kl-divergence-b0d57ee10e0a
- https://blog.keras.io/building-autoencoders-in-keras.html
- Kingma D P, Welling M. Auto-Encoding Variational Bayes[J]. stat, 2014, 1050: 10.
- DOERSCH C. Tutorial on Variational Autoencoders[J]. stat, 2016, 1050: 13.
- Ausif, Mahmood," Variational Autoencoders."Lecture of Deep Learning course from University of Bridgeport.
- Feifei,Li, "Generative Models."Lecture of CS231n: Convolutional Neural Networks for Visual Recognition from Stanford University.
- Blei, David M., "Variational Inference." Lecture of Advanced Methods in Probabilistic Modeling course from Princeton University.
- Kingma, D. P., Salimans, T., & Welling, M. (2015). Variational dropout and the local reparameterization trick. In Advances in Neural Information Processing Systems (pp. 2575-2583).