CPSC 440 Project Assignment

1. Name(s): Yilin Yang; Elvis Cai
2. Student ID(s): 24754350; 27910165

KL Divergence for two Gaussians

Recall the objective function for VAEs:

$$\arg\max_{q(z|x)} \; E_{q(z|x)}[log(p(x|z))] - KL(q(z|x)||p(z))$$

This is equivalently:

$$\operatorname{argmin}_{q(z|x)} -E_{q(z|x)}[log(p(x|z))] + KL(q(z|x)||p(z))$$

In the lecture, we introduced how KL divergence is used for VAEs, and emphasized how KL is utilized differently in Variational Inference and VAE. Here we will take a closer look:
The KL divergence formula for VI is as follows:

$$KL(q(z)||p(z|x)) = \int_{-\infty}^{\infty} q(z) log \frac{q(z)}{p(z|x)} dz$$

In VI, we approximate the posterior distribution $p(z|x)$ that is intractable with a rather tractable distribution. This finds a relatively good solution in a reasonable amount of time.
In VAE, notice how our KL portion of the objective function is actually:

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

In most VAE applications and for this question we make the assumption that $p(z) \sim N(0, I)$, i.e. we want the distribution of latent variable z to be as close to a standard normal distribution as possible. In addition and similar to most applications, we also assume $q(z|x)$ here is Gaussian.
Therefore the two distributions are both tractable and we are able to derive a closed form result.

1. Derive the closed form result of KL divergence between two univariate Gaussians, for simplicity, let $p(x) \sim N(\mu_1, \sigma_1)$ and $q(x) \sim N(\mu_2, \sigma_2)$.

   Answer: $KL(p(x)||q(x)) = \int q(x) log \frac{p(x)}{q(x)} dz = -\int p(x) \log q(x) dx + \int p(x) \log p(x) dx$
   For $\int p(x) \log p(x) dx$, from the normal distribution we have:
   $\int p(x) \log p(x) dx = \int p(x) \log \frac{1}{(2\pi\sigma_1^2)^{(1/2)}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} dx$
   $= -\frac{1}{2}\log(2\pi\sigma_1^2) + \int p(x)(-\frac{(x-\mu_1)^2}{2\sigma_1^2}) dx$
   $= -\frac{1}{2}\log(2\pi\sigma_1^2) - \frac{\int p(x)x^2 dx - \int p(x) 2x\mu_1 dx + \int p(x)\mu_1^2 dx}{2\sigma_1^2}$
   $= -\frac{1}{2}\log(2\pi\sigma_1^2) - \frac{E_p(x^2) - 2E_p(x)\mu_1 + \mu_1^2}{2\sigma_1^2}$
   $= -\frac{1}{2}\log(2\pi\sigma_1^2) - \frac{\sigma_1^2 - 2\mu_1^2 + \mu_1^2}{2\sigma_1^2}$
   $= -\frac{1}{2}(\log(2\pi\sigma_1^2) + 1)$
   Similarly, we can show that:

$\int p(x) \log p(x) dx = \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2}$ Therefore the KL divergence is: $KL(p(x)||q(x))$
$= \frac{1}{2}\log(2\pi\sigma_2^2) + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}(1 + \log(2\pi\sigma_1^2))$
$= \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$
Hint: For $\int p(x) \log p(x) dx$, only expand $\log p(x)$. Notice you can rewrite $\int p(x) x^k dx$ as $E_p(x^k)$ and that $Var_p(x) = E_p(x^2) - E_p(x)^2$ these will come in handy once you have expanded and simplified.

2. In practice the Multivariate Gaussians KL divergence is much more useful (we won't ask you to derive this) turns out that when $p(z) \sim N(0, 1)$,

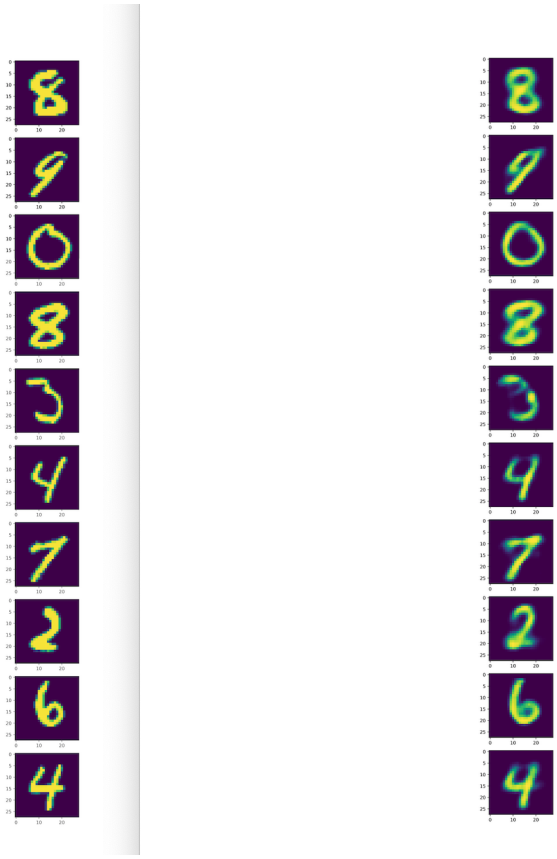
$$KL(q(z|x)||p(z)) = -\frac{1}{2}\sum_i [log(\sigma_i) + 1 - \sigma_i - \mu_i^2]$$

Running *example_ae.py* builds a deep autoencoder, which reconstruct the mnist digit data after compressing them into a latent variable of size 8. Implement a variational autoencoder model (fill in the missing functions and change the structure of the network), with the same amount of latent variables, and hand in your plot and code.

Answer: We can see that it performs better than the pure ae (Check *ae_answer.py*)

```
class autoencoder_util:

    def __init__(self):
        self.name = 'autoencoder'
        self.latent_dims=8
        return


    def flatten(self, x):
        x = x.astype('float32')/255
        x = x.reshape((len(x), np.prod(x.shape[1:])))
        return x



    def build_vae(self, training_shape):
        hidden_dims = [128,64]
        self.latent_dims = 10

        input_data = keras.Input(shape=(training_shape,))
        e1 = layers.Dense(hidden_dims[0],activation='relu')(input_data)
        e2 = layers.Dense(hidden_dims[1],activation='relu')(e1)

        latent_mean = layers.Dense(self.latent_dims, activation='relu')(e2)
        latent_log_cov = layers.Dense(self.latent_dims, activation='relu')(e2)

        latent = layers.Lambda(self.sampling)([latent_mean, latent_log_cov])

        d1 = layers.Dense(hidden_dims[1], activation = 'relu')(latent)
        d2 = layers.Dense(hidden_dims[0], activation = 'relu')(d1)
        output_data = Dense(training_shape, activation = 'sigmoid')(d2)

        vae = keras.Model(input_data, output_data)
        loss = self.get_vae_loss(latent_mean, latent_log_cov, input_data, output_data)
        vae.add_loss(loss)

        return vae

    def sampling(self, params):
        mean, log_cov = params
        sd_normal = K.random_normal(shape=(K.shape(mean)[0], self.latent_dims), mean = 0, stddev=0.1
        return mean + K.exp(log_cov)*sd_normal


    def get_vae_loss(self, mean, log_cov, input_data, output_data):
        ae_loss = 28*28*keras.losses.binary_crossentropy(input_data, output_data)
        kl_loss = -0.5*K.sum(1 + log_cov - K.square(mean) - K.exp(log_cov), axis=-1)
        return K.mean(ae_loss + kl_loss)
```

```
ae_util = autoencoder_util()

## Importing/Transforming mnist data
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = ae_util.flatten(x_train)
x_test = ae_util.flatten(x_test)
training_shape = x_train.shape[1]

## Build simple autoencoder model
vae = ae_util.build_vae(training_shape)
print(vae.summary())


## Compile and Fit model
vae.compile(optimizer='adam')
vae.fit(x_train, x_train, epochs=20, batch_size=256)

## Make predictions
pred = vae.predict(x_test)


## Plot predictions
sample_num = random.sample(range(0, len(x_test)), 10)
pred_out = pred[sample_num]
test_out = x_test[sample_num]

fig1, ax1 = plt.subplots(10,1)
fig2, ax2 = plt.subplots(10,1)

fig1.tight_layout()
fig2.tight_layout()


for x in range(0,10):
    test_img = np.reshape(test_out[x], (28,28))
    pred_img = np.reshape(pred_out[x], (28,28))
    ax1[x].imshow(test_img)
    ax2[x].imshow(pred_img)


fig1.set_figheight(30)
fig1.set_figwidth(30)
fig2.set_figheight(30)
fig2.set_figwidth(30)

fig1.savefig('vae_test.png', dpi=500)
fig2.savefig('vae_pred.png', dpi=500)
plt.show()
```



Hint: Some codes are commented out in the *ae.py*, which may help you with the implementation.