CS446: Machine Learning **Spring 2018**

Machine Problem 10

Handed Out: Jan. 16, 2018 *Due: Apr. 12, 2018 (11:59 AM Central Time)*

Note: The assignment will be autograded. It is important that you do not use additional libraries, or change the provided functions' input and output.

Part 1: Setup

- Connect to a EWS machine.

```
ssh (netid)@remlnx.ews.illinois.edu
```

- Load python module, this will also load pip and virtualenv. We use python 3.4 in this assignment.

```
module load python/3.4.3
```

- Reuse the virtual environment from the previous mps.

```
source ~/cs446sp_2018/bin/activate
```

- Go to your svn directory. Copy mp10 into your svn directory, and change directory to mp10.

```
cd ~/(netid)
svn cp https://subversion.ews.illinois.edu/svn/sp18-cs446/_shared/mp10 .
cd mp10
```

- Install the requirements through pip.

```
pip install --upgrade pip
pip install -r requirements.txt
```

- Create the data directory and prevent svn from checking in the data directory. Remember not to submit any training data to your svn directory.

```
mkdir MNIST_data
svn propset svn:ignore MNIST_data .
```

Part 2: Exercise

In this exercise, we will learn how to generate images from the data distribution using VAE. Review the lecture on Variational AutoEncoder for formulations. You are asked to implement a VAE.

Part 2.1 VAE

You are asked to implement a VAE to learn the data distribution of the MNIST dataset. For the ease of visualization, you can use 2-dimensional latent space. Use fully connected layers for both the encoder and decoder in the VAE. More details are provided in the function docstring.
In `main_vae.py`, the training procedure, data reader and visualization are provided for you. Fill in the functions in the file `vae.py`. To test your program, run `main_vae.py` and see the generated images stored in 'latent_space_vae.png'.

The visualized results will look something similar to the figure below.

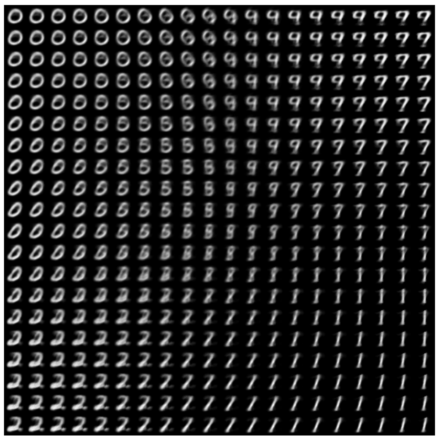
Figure 1: Generated images by VAE

Hints:

- You need to modify/implement the following functions in `vae.py`:

  1) _sample_z(self, z_mean, z_log_var): Sample z using the reparametrization trick given the outputs of the encoder: z_mean and z_log_var.

  2) _encoder(self, x): Build a two-layer network. Use two fully-connected layers with 100 nodes and then 50 nodes. Two output branches for the two (_nlatent)-dim vectors representing the z_mean and z_log_var.

  3) _decoder(self, z): Build a two-layer network. Use two fully-connected layers with 50 nodes and then 100 nodes. The output is a tensor containing the decoded images.

  4) _latent_loss(self, z_mean, z_log_var): Calculate the latent loss.

  5) _reconstruction_loss(self, f, x_gt): Calculate the reconstruction loss.

6) loss(self, f, x_gt, z_mean, z_var): The total loss of your VAE. You can call the above loss functions.

7) update_op(self, loss, learning_rate): Perform parameter update. New an Adam optimizer here using tf.train.AdamOptimizer to minimize the loss of your VAE and update the parameters.

8) generate_samples(self, z_np): Generate random samples from the provided latent samples z_np (z_np is a numpy array).

- You can use tensorflow.contrib.slim.fully_connected to implement the fully-connected layers in the network.
- The autograder will test your implementation in `vae.py` only. You can change `main_vae.py`.
- Remember to make the input dimension (ndims, the argument of VariationalAutoencoder's constructor) and the latent dimension (nlatent, the argument of VariationalAutoencoder's constructor) flexible. The auto-grader will test it.
- Make sure the reparameterization trick works correctly.
- Make sure the shape of the network follows the instruction.
- Make sure the losses are calculated correctly.
- The generated images should look similar to Figure 1.
- For your information, it took no more than one minute to train the VAE from scratch to get the result in Figure 1 in TA's implementation. We won't time your program, but you should keep this in mind.

Part 3: Writing Tests

In `test.py` we have provided basic test-cases. Make sure your code can pass the simple testing cases provided in test.py. Feel free to write more. To test the code, run

```
nose2
```

Part 4: Submit

Submit your python code to the svn repository. You don't have to submit the visualized images. Submitting the code is equivalent to committing the code. This can be done with the following command:

```
svn commit -m "Some meaningful comment here."
```

Lastly, double check on your browser. You can see your code at

```
https://subversion.ews.illinois.edu/svn/sp18-cs446/(netid)/mp10/
```

Make sure that the file `vae.py` is up-to-date in your svn repository by checking on browsers after you commit your code. **No late submission will be accepted.**