Train network

> The **network** tutorial simply compute the network output without any training process. Here, we are going to add a loss function and an optimizer to train the network.

```
import tensorflow as tf
import tensorflow.contrib.layers as Layers # my favorite. simple and
clean.
import numpy as np
import os

flags = tf.app.flags
flags.DEFINE_float('learning_rate', 1e-2, 'learning rate.')
flags.DEFINE_integer('epoch', 10, 'epoch to train model')
flags.DEFINE_string('save_path', 'ckpt', 'location to store models')
FLAGS = flags.FLAGS

def encoder(inputs):
    # 32x32x3 => 16x16x8
    h1 = Layers.conv2d(inputs, 8, 5, 2)
    # 16x16x8 => 8x8x16
    h2 = Layers.conv2d(h1, 16, 5, 2)
    # 8x8x16 => 4x4x16
    h3 = Layers.conv2d(h2, 16, 5, 2)
    # 4x4x16 => 256
    h3_flattened = tf.reshape(h3, [-1,256])
    # 256 => 2000
    h4 = Layers.fully_connected(h3_flattened,2000)
    # 2000 => 64
    h5 = Layers.fully_connected(h4, 64,activation_fn=None)
    return h5

def decoder(inputs):
    # 64 => 2000
    h1 = Layers.fully_connected(inputs, 2000)
    # 2000 => 256
    h2 = Layers.fully_connected(h1, 256)
    # 256 => 4x4x16
    h2 = tf.reshape(h2, [-1,4,4,16])
    # 4x4x16 => 8x8x16
    h3 = Layers.conv2d_transpose(h2, 16, 5, 2)
    # 8x8x16 => 16x16x8
    h4 = Layers.conv2d_transpose(h3, 8, 5, 2)
    # 16x16x8 => 32x32x3
    h5 = Layers.conv2d_transpose(h4, 3, 5, 2, activation_fn=tf.sigmoid)
    return h5

def network(inputs):
    code = encoder(inputs)
    outputs = decoder(code)
```

```
    return outputs

def main(_):
    # NOTE: this is a placeholder (like a box to contain future inputs)
    inputs = tf.placeholder(tf.float32, [10, 32, 32, 3])
    # build the network
    outputs = network(inputs)
    # compute the loss function.
    loss = tf.reduce_mean(tf.square(inputs - outputs))

    # optimizer
    optim = tf.train.AdamOptimizer(FLAGS.learning_rate)
    train_op = optim.minimize(loss)

    # create a session
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        # random generate fake images
        imgs = np.random.random(size=[10,32,32,3])

        # train model for a number of epochs
        saver = tf.train.Saver()
        for i in range(FLAGS.epoch):
            sess.run(train_op, feed_dict={inputs:imgs})
            loss_value = sess.run(loss, feed_dict={inputs:imgs})
            print(loss_value) # print loss

            # save model
            saver.save(sess,
os.path.join(FLAGS.save_path,'model'),global_step=i)


if __name__ == '__main__':
    tf.app.run()
```

Things to be noticed:

1. *tf.app_flags*

   A command line option. you can use the program by typing the following in the terminal.

   ```
   python train.py --learning_rate=0.0002 --save_path=my_save_model
   ```

   Note: you have to use *tf.app_flags* together with *tf.app.run*

2. optimizer

   A number of optimizers can be used. I only use adam as demonstration. The simplest way to optimize your model is to directly minimize some loss function. Other usage might be covered in other parts

3. saver

   You can save your model by calling saver. The default usage is to save all variables used in the program.