PART 2 TensorFlow

3. Workshop 1 - CNN 影像處理基礎

REFERENCE

1. Tom Hope, Yehezkel S. Resheff, Itay Lieder, "**Learning TensorFlow - A Guide to building Deep Learning Systems**", `Chapters 2 & 4`, O'Reilly (2017) (pdf) https://goo.gl/iEmehh (https://goo.gl/iEmehh) `[ Code ]` : https://github.com/gigwegbe/Learning-TensorFlow (https://github.com/gigwegbe/Learning-TensorFlow)
2. bigDataSpark Forum 檔案：**Basics of TensorFlow Programming-20180809.ipynb** https://www.facebook.com/groups/753114451505938/permalink/1213353432148 (https://www.facebook.com/groups/753114451505938/permalink/121335343214

Convolution

With convolutional neural networks, we use the built-in TensorFLow conv2d( ) :

```
tf.nn.conv2d(x, w, strides=[1,1,1,1], padding ='SAME')
```

- The strides argument controls the spatial movement of the filter W across the image (or feature map) x.
- The value `[1,1,1,1]` means that the filter is applied to the input in one-pixel intervals in each dimension, corresponding to a 'full' convolution.
- Finally, the setting padding to `"SAME"` means that the borders of x padded such that the size of the result of the operation is the same as the size of x.

[NOTE] : x is the data (input image). `Feature map` is simply a commonly used term referring to the output of each layer. The output of this operation will depend on the shape of x and W, in this case is four-dimensional.

Starting TensorFlow ...

In [1]:

```
import tensorflow as tf
import numpy as np
```

```
/Users/macmini1/anaconda3/lib/python3.6/site-packages/h5p
y/__init__.py:36: FutureWarning: Conversion of the second
argument of issubdtype from `float` to `np.floating` is de
precated. In future, it will be treated as `np.float64 ==
np.dtype(float).type`.
  from ._conv import register_converters as _register_conv
erters
```

Building a Graph ...

In [2]:

```
def weight_variable(shape):
    """ This specifies the weights for either fully connected or co
       of the network. They are initialized randomly using a trunca
       with a standard deviation of .1. """
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    """ This defines the bias elements in either a fully connected
        These are all initialized with the constant value of .1."""
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def conv2d(x, W):
    """ This specifies the convolution we will typically use. A ful
        with an output the same size as the input."""
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    """ This sets the max pool to half the size across the height/w
        and in total a quarter the size of the feature map."""
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                          strides=[1, 2, 2, 1], padding='SAME')

def conv_layer(input, shape):
    """ This is the actual layer we will use. Linear convolution as
        followed by the ReLU nonlinearity."""
    W = weight_variable(shape)
    b = bias_variable([shape[3]])
    return tf.nn.relu(conv2d(input, W) + b)

def full_layer(input, size):
    """ A standard full layer with a bias. Notice that here we didn
        This allows us to use the same layer for the final output,
        where we don't need the non-linear part."""
    in_size = int(input.get_shape()[1])
    W = weight_variable([in_size, size])
    b = bias_variable([size])
    return tf.matmul(input, W) + b
```

1. Tensors (Ref. 1 : Fig. 4-4)

In [3]:

```
##  Defining the placeholders for the images and correct labels,
##  x and y_, respectively.
x = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 10])

##  Reshape the image data into the 2D image format with size 28×28
##   -1 : 代表自動計算該維度的數量
##    1 : 代表 1 channel for MNIST dataset (greyscale)
x_image = tf.reshape(x, [-1, 28, 28, 1])

##-------------------------------------------------------------------
##  << Deep Network Design >> :  (Ref. 1 : Fig. 4-4)
##     Two consecutive layers of convolution and pooling,
##     each with 5×5 convolutions and 64 feature maps,
##     followed by a single fully connected layer with 1,024 units.
##-------------------------------------------------------------------

##  ---------------------------------------------------
##  Conv_Layer 1    :  32 filters (5x5 & 1 channel)
##  Pooling_Layer 1 :  max_pool (2x2, strides = 2x2)
##  ---------------------------------------------------

conv1 = conv_layer(x_image, shape=[5, 5, 1, 32])
conv1_pool = max_pool_2x2(conv1)    ##  The input 28×28 pixel image

##  The size of one image after these two convolution and pooling l

##  ---------------------------------------------------
##  Conv_Layer 2    :  64 filters (5x5 & 32 feature maps)
##  Pooling_Layer 2 :  max_pool (2x2, strides = 2x2)
##  ---------------------------------------------------

conv2 = conv_layer(conv1_pool, shape=[5, 5, 32, 64])
conv2_pool = max_pool_2x2(conv2)    ##  The input 14×14 pixel image

##  The size of one image after the second convolution and pooling

##  ---------------------------------------------------------------
##  [Input data for the Fully-connected Network] :
##    Flattern the 64 feature maps (each with size 7x7),
##    i.e., 7x7x64 = 3136 input data for the fully-connected networ
##  ---------------------------------------------------------------

conv2_flat = tf.reshape(conv2_pool, [-1, 7*7*64])
full_1 = tf.nn.relu(full_layer(conv2_flat, 1024))  ##  Output-data

##  Dropout for regularization in order to prevent overfitting...
##  [ The parameter 'keep_prob' ] :
##     - is the fraction of the neurons to keep working at each ste
##     - if 'keep_prob' = 1.0, it means no dropout at all.

keep_prob = tf.placeholder(tf.float32)
full1_drop = tf.nn.dropout(full_1, keep_prob=keep_prob)

y_conv = full_layer(full1_drop, 10)  ##  Output Layer : size = 10
```

2. Input MNIST dataset (Ref. 1 : Chapter 2)

In [4]:

```
# for the old-version usage of TensorFlow, such as tensorflow.examp
old_v = tf.logging.get_verbosity()
tf.logging.set_verbosity(tf.logging.ERROR)

##  Loading the input data, MNIST  (Ref. 1 : Chapter 2)
from tensorflow.examples.tutorials.mnist import input_data

DATA_DIR = './data'
STEPS = 1000
MINIBATCH_SIZE = 50

##  The parameter 'one_hot' : setting the labelled data with 1 and
mnist = input_data.read_data_sets(DATA_DIR, one_hot=True)

mnist
```

```
Extracting ./data/train-images-idx3-ubyte.gz
Extracting ./data/train-labels-idx1-ubyte.gz
Extracting ./data/t10k-images-idx3-ubyte.gz
Extracting ./data/t10k-labels-idx1-ubyte.gz
```

Out[4]:

```
Datasets(train=<tensorflow.contrib.learn.python.learn.data
sets.mnist.DataSet object at 0x12237a048>, validation=<ten
sorflow.contrib.learn.python.learn.datasets.mnist.DataSet
object at 0x1222e26a0>, test=<tensorflow.contrib.learn.pyt
hon.learn.datasets.mnist.DataSet object at 0x1222e2630>)
```

In [5]:

```
print(" mnist.train.images.shape :\t ", mnist.train.images.shape)
print(" mnist.train.labels.shape :\t ", mnist.train.labels.shape)
print(" mnist.validation.images.shape : ", mnist.validation.images.
print(" mnist.validation.labels.shape : ", mnist.validation.labels.
print(" mnist.test.images.shape :\t ", mnist.test.images.shape)
print(" mnist.test.labels.shape :\t ", mnist.test.labels.shape)

mnist.test.labels[0]
```

```
 mnist.train.images.shape :       (55000, 784)
 mnist.train.labels.shape :       (55000, 10)
 mnist.validation.images.shape :  (5000, 784)
 mnist.validation.labels.shape :  (5000, 10)
 mnist.test.images.shape :        (10000, 784)
 mnist.test.labels.shape :        (10000, 10)
```

Out[5]:

```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.])
```

3. Optimization

In [6]:

```
##  ------------------------------------------------------------------
## [ Activation Function for the prediction ] - Softmax
##
## [ Loss Function ] : using Cross Entropy
##    - Cross entropy is a natural choice when the model outputs cl
##    - This element is often referred to as the loss function.
##  ------------------------------------------------------------------

cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_log

##  ------------------------------------------------------------------
##  Using Adam algorithm (with learning rate = 1e-4) for the optimi
##  ------------------------------------------------------------------

train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

4. Accuracy

In [7]:

```
##  ------------------------------------------------------------------
##  Computing the prediction ...
##
##     tf.argmax(input, axis=NONE, ...)
##        - Returns the index with the largest value across axes of
##        -  axis = 0 : across a row
##        -  axis = 1 : across a column.
##  ------------------------------------------------------------------

correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1


##  ------------------------------------------------------------------
##  Computing the accuracy ...
##  ------------------------------------------------------------------

accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Launch the Graph

In [11]:

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for i in range(STEPS):
        batch = mnist.train.next_batch(MINIBATCH_SIZE)  ## MINIBATC

        if (i+1)%100 == 0:
            train_accuracy = sess.run(accuracy,
                                      feed_dict={x: batch[0],
                                                 y_: batch[1],
                                                 keep_prob: 1.0})
            print("[ STEP {} ] :\t Training Accuracy = {}".format(s

        sess.run(train_step, feed_dict={x: batch[0], y_: batch[1],

    print("\n Computing the test accuracy ... ", end = " ")

    ##  ----------------------------------------------------------
    ##  Split the test procedure into 10 blocks of 1,000 images eac
    ##  Doing this is important mostly for much larger datasets.
    ##  ----------------------------------------------------------

    X_test = mnist.test.images.reshape(10, 1000, 784)  ##  mnist.te
    Y_test = mnist.test.labels.reshape(10, 1000, 10)   ##  mnist.te

    test_accuracy = np.mean([sess.run(accuracy,
                                      feed_dict={x:X_test[i], y_:Y_
                                      for i in range(10)])
    print(" Done !!! ")
```

```
[ STEP  100 ] :  Training Accuracy = 0.8399999737739563
[ STEP  200 ] :  Training Accuracy = 0.8199999928474426
[ STEP  300 ] :  Training Accuracy = 0.9599999785423279
[ STEP  400 ] :  Training Accuracy = 0.9800000190734863
[ STEP  500 ] :  Training Accuracy = 0.8999999761581421
[ STEP  600 ] :  Training Accuracy = 0.9800000190734863
[ STEP  700 ] :  Training Accuracy = 0.9399999976158142
[ STEP  800 ] :  Training Accuracy = 0.9399999976158142
[ STEP  900 ] :  Training Accuracy = 0.9800000190734863
[ STEP 1000 ] :  Training Accuracy = 0.9399999976158142

 Computing the test accuracy ...   Done !!!
```

Output the test accuracy ...

In [12]:

```
print("\n [ Test Accuracy ] : {}".format(test_accuracy))
```

```
 [ Test Accuracy ] : 0.960800051689148
```

[EXERCISE 1] :

上述程式範例中，請增加 Fully-Connected Deep Networks 的隱藏層，計算並繪製 training & validation curves。

> [Hint] : 請參考 PART 2 **`1. TensorFlow 程式設計基礎`** 一章的內容與程式！

[EXERCISE 2] :

請將上述程式範例，輸出結果至 TensorBoard。

> [Hint] : Using **`tf.name_scope`** , **`tf.summary`** , ...

In [ ]:

```
1
```