PART 2 TensorFlow

1. TensorFlow 程式設計基礎

C. Alex Hu

Import TensorFlow & run a version check...

In [1]:

```
1 import tensorflow as tf
2 print(tf.__version__)
```

```
/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
```

```
1.12.0
```

1. Build a graph...

tf.constant

```
tf.constant(
    value,
    dtype=None,
    shape=None,
    name='Const',
    verify_shape=False_
)
```

- Creates a constant tensor.
- The resulting tensor is populated with values of type *dtype*, as specified by arguments *value* and (optionally) *shape* (see examples below).
- https://www.tensorflow.org/api_docs/python/tf/constant (https://www.tensorflow.org/api_docs/python/tf/constant)

In [2]:

```
# Build a graph...
h  = tf.constant("Hello")
TF = tf.constant("TensorFlow!")
hTF = h + ' ' + TF

# Constant 1-D Tensor populated with value list.
fibo = tf.constant([1, 1, 2, 3, 5, 8, 13, 21])

# Constant 3-D tensor populated with scalar value 1.0.
allOneMatrix = tf.constant(1.0, shape=[3, 3])
```

2. Launch the graph in a session...

tf.Session

- Class **Session** - A class for running TensorFlow operations.
  https://www.tensorflow.org/api_docs/python/tf/Session (https://www.tensorflow.org/api_docs/python/tf/Session)
- Defined in `tensorflow/python/client/session.py` :
  - https://github.com/tensorflow/tensorflow/blob/r1.10/tensorflow/python/client/s (https://github.com/tensorflow/tensorflow/blob/r1.10/tensorflow/python/client/s
- See the guides: Running Graphs > Session management, Running Graphs
  - https://www.tensorflow.org/api_guides/python/client (https://www.tensorflow.org/api_guides/python/client)

- A *Session* object encapsulates the environment in which *Operation* objects are executed, and *Tensor* objects are evaluated.
- A session may own resources, such as **tf.Variable**, **tf.QueueBase**, and **tf.ReaderBase**. It is important to release these resources when they are no longer required.
- To do this, either invoke the **tf.Session.close** method on the session, or use the session as a context manager.

< Example 1 > Using the `close()` method.

In [3]:

```
##  < Example 1 > Using the `close()` method.
##
# Launch the graph in a session.
sess = tf.Session()

# Evaluate the tensor `c`.
print(sess.run(hTF), end='\n\n')
print('Fibonacci numbers = ', sess.run(fibo), end='\n\n')
print('All-one 3D Tensor : \n', sess.run(allOneMatrix))

# Using the `close()` method.
sess.close()
```

```
b'Hello TensorFlow!'

Fibonacci numbers =  [ 1  1  2  3  5  8 13 21]

All-one 3D Tensor :
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

< Example 2 > Using the context manager.

In [4]:

```
##  < Example 2 > Using the context manager.
##
# Launch the graph in another session by using the context manager.
with tf.Session() as sess:
    azm = sess.run(allOneMatrix)
    print(sess.run(hTF), end='\n\n')
    print('Fibonacci numbers = ', end='')
    print(sess.run(fibo), end='\n\n')

print('All-One 3D Tensor : \n', azm)
```

```
b'Hello TensorFlow!'

Fibonacci numbers = [ 1  1  2  3  5  8 13 21]

All-One 3D Tensor :
 [[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

- **Outputs in TensorFlow...**

In [5]:

```
print(hTF)  #  Output in TensorFlow
```

```
Tensor("add_1:0", shape=(), dtype=string)
```

In [6]:

```
print(allOneMatrix)  #  Output in TensorFlow
```

```
Tensor("Const_3:0", shape=(3, 3), dtype=float32)
```

In [7]:

```
print(fibo)  #  Output in TensorFlow
```

```
Tensor("Const_2:0", shape=(8,), dtype=int32)
```

`run` method

```
run(
    fetches,
    feed_dict=None,
    options=None,
    run_metadata=None
)
```

- Runs operations and evaluates tensors in `fetches` .
- The `fetches` argument may be a single graph element, or an arbitrarily nested list, tuple, namedtuple, dict, or OrderedDict containing graph elements at its leaves.
- Ref : https://www.tensorflow.org/api_docs/python/tf/Session (https://www.tensorflow.org/api_docs/python/tf/Session)

- **The session object may call the `run` mehod to act as an interface to run parts of the computation graph externally. For example :**

In [8]:

```
with tf.Session() as sess:
    fibo_num = sess.run(fibo)

print(fibo_num)
```

```
[ 1  1  2  3  5  8 13 21]
```

In [9]:

```
import collections
MyData = collections.namedtuple('MyData', ['a', 'b'])

with tf.Session() as sess:
    s = tf.constant('S')
    a = tf.constant([10, 20])
    b = tf.constant([1.0, 2.0])
    v = sess.run(s)       # 'fetches' can be a singleton
    print(v, end='\n\n')
    v = sess.run(a)       # v is the numpy array [10, 20] => 'fetch
    print(v, end='\n\n')
    v = sess.run([a, b])  # v is a Python list with 2 numpy arrays:
    print(v, end='\n\n')

   # 'fetches' can be arbitrary lists, tuples, namedtuple, dicts:
    v = sess.run({'k1': MyData(a, b), 'k2': [b, a]})
    print(v, end='\n\n')
   # v is a dict with
   # v['k1'] is a MyData namedtuple with 'a' (the numpy array [10,
   # 'b' (the numpy array [1.0, 2.0])
   # v['k2'] is a list with the numpy array [1.0, 2.0] and the nump
   # [10, 20].

print(v)
```

```
b'S'

[10 20]

[array([10, 20], dtype=int32), array([1., 2.], dtype=float
32)]

{'k1': MyData(a=array([10, 20], dtype=int32), b=array([1.,
2.], dtype=float32)), 'k2': [array([1., 2.], dtype=float3
2), array([10, 20], dtype=int32)]}

{'k1': MyData(a=array([10, 20], dtype=int32), b=array([1.,
2.], dtype=float32)), 'k2': [array([1., 2.], dtype=float3
2), array([10, 20], dtype=int32)]}
```

3. Create placeholders & variables ...

tf.placeholder

```
tf.placeholder(
    dtype,
    shape=None,
    name=None
)
```

- Inserts a placeholder for a tensor that will be always fed.
- Defined in `tensorflow/python/ops/array_ops.py` :
  - https://github.com/tensorflow/tensorflow/blob/r1.10/tensorflow/python/ops/arr (https://github.com/tensorflow/tensorflow/blob/r1.10/tensorflow/python/ops/ar
- See the guides: Inputs and Readers > Placeholders
  - https://www.tensorflow.org/api_guides/python/io_ops#Placeholders (https://www.tensorflow.org/api_guides/python/io_ops#Placeholders)

- NOTE : This tensor will produce an error if evaluated. Its value must be fed using the `feed_dict` optional argument to `Session.run()` , `Tensor.eval()` , or `Operation.run()` .

In [10]:

```
# Build a graph...
import numpy as np
x = tf.placeholder(tf.float32, shape=(1024, 1024))
y = tf.matmul(x, x)

# Launch the built graph in a session...
with tf.Session() as sess:
  ## print(sess.run(y))  # ERROR: will fail because x was not fed.

  rand_array = np.random.rand(1024, 1024)
  print(sess.run(y, feed_dict={x: rand_array}))  # Will succeed.
```

```
[[251.51971 249.96445 249.7326  ... 249.05339 254.70279 24
9.6082 ]
 [258.1219  250.57448 255.17242 ... 251.07181 260.55475 25
3.86662]
 [267.00653 264.4317  268.29324 ... 255.58131 271.63013 26
3.03745]
 ...
 [262.60315 256.61667 260.42203 ... 257.08786 267.29364 25
5.07413]
 [255.45755 248.70311 252.51103 ... 247.57643 261.73187 25
4.90831]
 [256.54367 248.86333 257.74802 ... 247.29007 264.3275  25
6.78043]]
```

tf.Variable

- Class **Variable** https://www.tensorflow.org/api_docs/python/tf/Variable (https://www.tensorflow.org/api_docs/python/tf/Variable)
- Defined in `tensorflow/python/ops/variables.py` :
  - https://github.com/tensorflow/tensorflow/blob/r1.10/tensorflow/python/ops/va (https://github.com/tensorflow/tensorflow/blob/r1.10/tensorflow/python/ops/va
- See the guides: Variables > Variables
  - https://www.tensorflow.org/api_guides/python/state_ops#Variables (https://www.tensorflow.org/api_guides/python/state_ops#Variables)

- A variable maintains state in the graph across calls to **`run()`** method. You add a variable to the graph by constructing an instance of the class Variable.
- The **`Variable()`** constructor requires an initial value for the variable, which can be a `Tensor` of any type and shape.

In [11]:

```
# Build a graph...
# Create two variables.
w = tf.Variable(tf.random_normal([2, 3], stddev=1, seed=1))
b = tf.Variable(tf.random_normal([1, 3], stddev=1, seed=1))

# Generate a placeholder.
x = tf.placeholder(tf.float32, shape=(None, 2), name='input')

# Use the variables in the graph like any Tensor.
a = tf.matmul(x, w)               #  tf.matmul(x, w)  =>  a = x * w
z = tf.add(a, b)                  #  tf.add(a, b)  =>  a + b
y = tf.sigmoid(z)                 #  Computing Sigmoid Function...
d = tf.constant([[0.35, 0.8]])    #  a 2d tensor for x : 1x2

# Launch the built graph in a session...
with tf.Session() as sess:
    sess.run(w.initializer)
    sess.run(b.initializer)
    print(' w : \n', sess.run(w))
    print(' x = ', sess.run(d))
    print('\n a = w * x =  ', sess.run(a, feed_dict={x: sess.run(d)
    print('\t b =  ', sess.run(b))
    zi = sess.run(z, feed_dict={x: sess.run(d)})
    print('\n z = w*x + b = ', zi)
    print('\n y = Sigmoid(z) = ', sess.run(y, feed_dict={x: sess.ru
```

```
w :
[[-0.8113182   1.4845988   0.06532937]
[-2.4427042   0.0992484   0.5912243 ]]
x =  [[0.35 0.8 ]]

a = w * x =   [[-2.2381248   0.5990083   0.49584472]]
        b =   [[-0.8113182   1.4845988   0.06532937]]

z = w*x + b =  [[-3.049443   2.0836072  0.5611741]]

y = Sigmoid(z) =  [[0.04524153 0.88929963 0.6367242 ]]
```

Q : How to check if the answer above is correct or not?

[Hint] :

- `from math import exp`
- **Sigmoid Function :**
  - $\sigma(z) = 1 / [1 + exp(-z)]$

In [12]:

```
1 import numpy as np
2 sig = [1.0 / (1 + np.exp(-i)) for i in zi]
3 sig
```

Out[12]:

```
[array([0.04524153, 0.88929963, 0.6367242 ], dtype=float3
2)]
```

[Further Reading about Computing Graph] - `Tensor-Flow` ...

- Tom Hope, Yehezkel S. Resheff, and Itay Lieder, "Learning TensorFlow : A Guide to Building Deep Learning Systems," Chapter 2, O'Reilly, 2017.

[Project] : Hand-written Digits Recognition - `MNIST` dataset

Reference :

- Tom Hope, Yehezkel S. Resheff, and Itay Lieder, "**Learning TensorFlow : A Guide to Building Deep Learning Systems**," Chapter 2, Example 2-2, O'Reilly, 2017. https://goo.gl/iEmehh (https://goo.gl/iEmehh)
- Download the code from GitHub : https://github.com/gigwegbe/Learning-TensorFlow (https://github.com/gigwegbe/Learning-TensorFlow)

In [13]:

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

#import MNIST dataset
from tensorflow.examples.tutorials.mnist import input_data
```

Loading the MNIST datasets

- The second argument, **one_hot=True** , shows the data to be labeled with **one-hot encoding**.

In [15]:

```
DATA_DIR = "./data"
mnist = input_data.read_data_sets(DATA_DIR, one_hot=True)
```

```
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
```

- Checking the dataset...

In [16]:

```
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.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)
```

In [17]:

```
print('mnist.train.images[0] : \n', \
      ' min = ', min(mnist.train.images[0]), \
      ' max = ', max(mnist.train.images[0]))
```

```
mnist.train.images[0] :
  min =  0.0  max =  0.9960785
```

In [18]:

```
mnist.train.labels[0]
```

Out[18]:

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

Setting the parameters

In [19]:

```
# Initializing some parameters...
# Hyperparameters
Learning_Rate = 0.5
NUM_STEPS = 1000
MINIBATCH_SIZE = 128
Display_Step = 100

# Network Parameters
Node_hidden_1 = 256    # Number of Neurons in Hidden layer 1
Node_hidden_2 = 128    # Number of Neurons in Hidden layer 2
Node_Inputs = 784      # Number of MNIST data input (Each image shap
Node_Outputs = 10      # Number of output classes (digits 0 ~ 9)
```

Building a graph...

< Forward Propagation >

- The image, **X** : a placeholder, supplied when running the computation graph.
- The size **[None, 784]** :
  - **784** (= 28x28 pixels) is the size of each image,
  - **None** means not currently specifying the number of these images used each time.

In [20]:

```
# Each Input Image, X, with 28*28 (= 784) pixels
X = tf.placeholder(tf.float32, [None, Node_Inputs])

# y_true : the training labeled dataset
y_true = tf.placeholder(tf.float32,[None, Node_Outputs])
```

In [21]:

```
# Initializing Weights & Biases for Nodes in All Hidden Layers
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)
```

- **Building a Fully-Connected Deep Network with 2 Hidden Layers**
  - Model Outputs for Prediction without Activation-Function Processing

In [22]:

```
# Building a Fully-Connected Deep Network
def full_layer(inputs, size):
    in_size = int(inputs.get_shape()[1])
    W = weight_variable([in_size, size])
    b = bias_variable([size])
    return tf.add(tf.matmul(inputs, W), b)
```

In [23]:

```
##  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)

# < Hidden Layer 1 >
layer_1_drop = tf.nn.dropout(X, keep_prob=keep_prob)
#   Activation Function : ReLU
layer_1_Outputs = tf.nn.relu(full_layer(layer_1_drop, Node_hidden_1

# < Hidden Layer 2 >
layer_2_drop = tf.nn.dropout(layer_1_Outputs, keep_prob=keep_prob)
#   Activation Function : ReLU
layer_2_Outputs = tf.nn.relu(full_layer(layer_2_drop, Node_hidden_2

# < Output Layer >
output_drop = tf.nn.dropout(layer_2_Outputs, keep_prob=keep_prob)
# Without Activation Function
y_pred = full_layer(output_drop, Node_Outputs)
```

< Back-propagation >

- Activation Function for **`Model-Output Prediction`** : **Softmax**
  - `tf.nn.softmax_cross_entropy_with_logits(logits=y_pred, labels=y_true)`
- Loss function : **Cross entropy with logits**
- Optimizer : **Gradient Descent** with the method **`minimize()`**
  - [Note] : 0.5 is the learning rate
- Computing the **Accuracy Score**

In [24]:

```
# Computing the loss scores with Categorical Cross Entropy
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_log

# Gradient Descent Optimizer with Learning_Rate = 0.5
gd_step = tf.train.GradientDescentOptimizer(Learning_Rate).minimize

# Computing Accuracy Scores...
correct_mask = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y_true, 1))
accuracy = tf.reduce_mean(tf.cast(correct_mask, tf.float32))
```

Launching the graph...

In [25]:

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

    # Training Process...
    sess.run(tf.global_variables_initializer())

    # Outputs for the history of training & validation
    loss = []
    accu = []
    val_loss = []
    val_accu = []

    ##  mnist.test.validation.shape : (5000, 784)
    X_val = mnist.validation.images.reshape(5, 1000, 784)
    ##  mnist.test.validation.shape : (5000, 10)
    Y_val = mnist.validation.labels.reshape(5, 1000, 10)

    for i in range(NUM_STEPS):
        batch_xs, batch_ys = mnist.train.next_batch(MINIBATCH_SIZE)
        sess.run(gd_step, feed_dict ={X: batch_xs,
                                      y_true: batch_ys,
                                      keep_prob: 0.5})

        if (i+1) % Display_Step == 0:
            # Calculate batch loss and accuracy
            loss_temp, accu_temp = sess.run([cross_entropy, accurac
                                             feed_dict={X: batch_xs,
                                                        y_true: batc
                                                        keep_prob: 1
            loss.append(loss_temp)
            accu.append(accu_temp)

        # Validating Process...
            v_loss, v_accu = 0., 0.
            for j in range(5):
                v_l, v_a = sess.run([cross_entropy, accuracy],
                                             feed_dict={X: X_val[j],
                                                        y_true: Y_va
                                                        keep_prob: 1
                v_loss += v_l
                v_accu += v_a
            val_loss.append(v_loss/5)
            val_accu.append(v_accu/5)

            print("Step " + str(i+1).rjust(4) + \
                  " : Loss = " + "{:.4f}".format(loss_temp) + \
                  ", Accuracy = " + "{:.3f}".format(accu_temp) + \
                  " ; Val_Loss = " + "{:.4f}".format(v_loss/5) + \
                  ", Val_Accuracy = " + "{:.3f}".format(v_accu/5))

    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.
    ##  ------------------------------------------------------------
    ##  mnist.test.images.shape : (10000, 784)
    X_test = mnist.test.images.reshape(10, 1000, 784)
```

```
    ##  mnist.test.labels.shape : (10000, 10)
    Y_test = mnist.test.labels.reshape(10, 1000, 10)

    test_loss = np.mean([sess.run(cross_entropy,
                                  feed_dict={X: X_test[i],
                                             y_true: Y_test[i],
                                             keep_prob: 1.0})
                                  for i in range(10)])
    test_accu = np.mean([sess.run(accuracy,
                                  feed_dict={X: X_test[i],
                                             y_true: Y_test[i],
                                             keep_prob: 1.0})
                                  for i in range(10)])
    print(" Done !!! ")
```

```
Step  100 : Loss = 0.5917, Accuracy = 0.859 ; Val_Loss =
0.5638, Val_Accuracy = 0.857
Step  200 : Loss = 0.4202, Accuracy = 0.867 ; Val_Loss =
0.4437, Val_Accuracy = 0.879
Step  300 : Loss = 0.3778, Accuracy = 0.891 ; Val_Loss =
0.3584, Val_Accuracy = 0.910
Step  400 : Loss = 0.2490, Accuracy = 0.938 ; Val_Loss =
0.3122, Val_Accuracy = 0.919
Step  500 : Loss = 0.2783, Accuracy = 0.906 ; Val_Loss =
0.2768, Val_Accuracy = 0.929
Step  600 : Loss = 0.1986, Accuracy = 0.969 ; Val_Loss =
0.2383, Val_Accuracy = 0.936
Step  700 : Loss = 0.2301, Accuracy = 0.953 ; Val_Loss =
0.2283, Val_Accuracy = 0.941
Step  800 : Loss = 0.1602, Accuracy = 0.969 ; Val_Loss =
0.2184, Val_Accuracy = 0.941
Step  900 : Loss = 0.1869, Accuracy = 0.938 ; Val_Loss =
0.2069, Val_Accuracy = 0.941
Step 1000 : Loss = 0.2198, Accuracy = 0.922 ; Val_Loss =
0.2043, Val_Accuracy = 0.942

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

In [26]:

```
print(" [ Test  Accuracy ] : {}".format(test_accu) +
      "\n [ Test Loss Score ] : {}".format(test_loss))
```

```
[ Test  Accuracy ] : 0.9387999773025513
[ Test Loss Score ] : 0.211587592959404
```

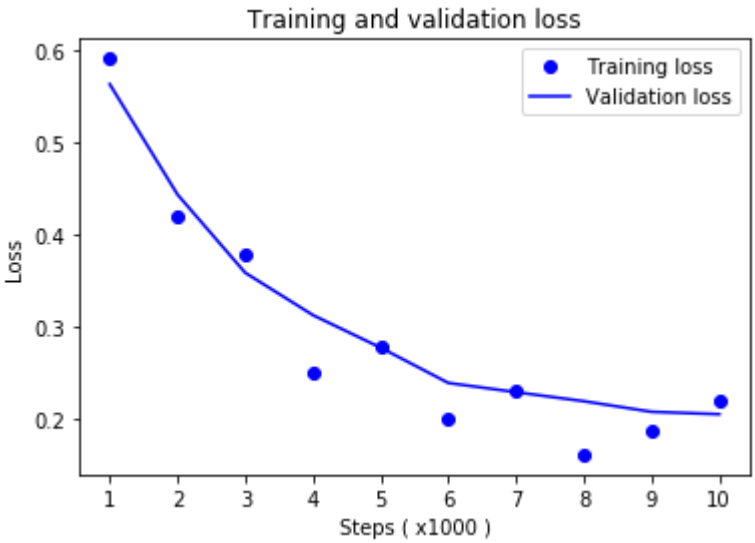
In [27]:

```
import matplotlib.pyplot as plt
%matplotlib inline

steps = range(1, len(accu) + 1)

# "bo" is for "blue dot"
plt.plot(steps, loss, 'bo', label='Training loss')
# b is for "solid blue line"
plt.plot(steps, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Steps ( x1000 )')
plt.ylabel('Loss')
plt.xticks(steps)
plt.legend()

plt.show()
```

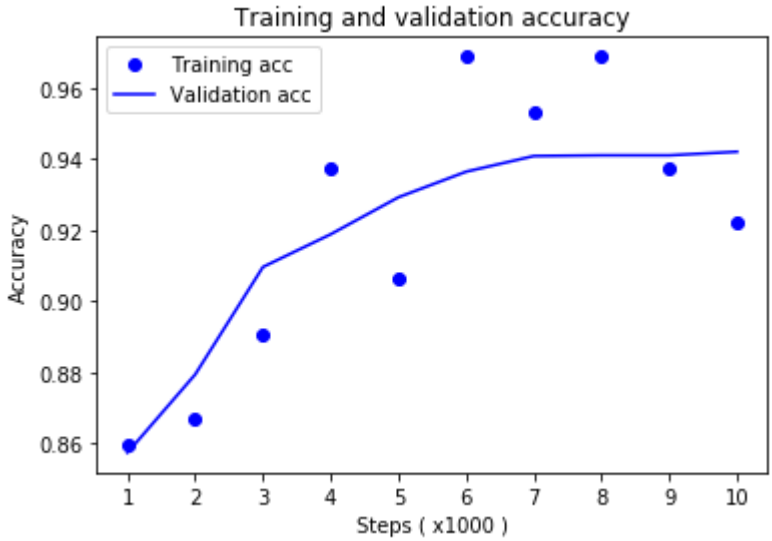


In [28]:

```
plt.clf()   # clear figure

plt.plot(steps, accu, 'bo', label='Training acc')
plt.plot(steps, val_accu, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Steps ( x1000 )')
plt.ylabel('Accuracy')
plt.xticks(steps)
plt.legend()

plt.show()
```



Q : Try to increasing more steps or rerunning the training-and-testing process for the example above. What happens?

Two useful websites from Google :

- **[Colaboratory] :**
  https://colab.research.google.com/notebooks/welcome.ipynb (https://colab.research.google.com/notebooks/welcome.ipynb)
- **[Google Codelabs 學習網站] : TensorFlow and deep learning, without a PhD**
  https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0 (https://codelabs.developers.google.com/codelabs/cloud-tensorflow-mnist/#0)