Tensorflow

Ziwei Bai

- TensorFlow enables researchers to build machine learning model
- Combination of strategy and mechanism

1、 Using the **graph** to represent the computational task
2、 Launch the graph in **session**
3、 Using **tensor** to represent data
4、 Using **variable** maintenance state
5、 Using **feed** and **fetch** to assign for and get data from any operation

Process

1、 Build a graph
2、 Launch the graph in a session
3、 close the session

https://www.tensorflow.org/

```
# Build a graph.
a = tf.constant(5.0)
b = tf.constant(6.0)
c = a * b
# Launch the graph in a session.
sess = tf.Session()
print sess.run(c)
#close session
Sess.close()
```

Constant Variable placeholder

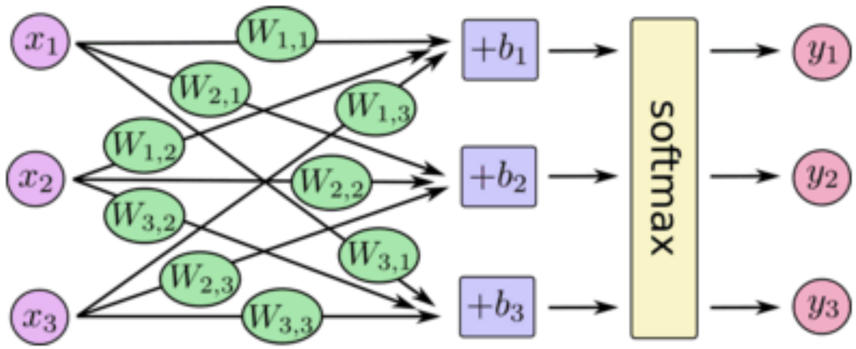
Constant：
tf.constant(value, dtype=None, shape=None, name='Const')

Variable：
vs.get_variable(name, shape=None, dtype=tf.float32, initializer=None, regularizer=None, trainable=True, collections=None)

Placeholder
placeholder(dtype, shape=None, name=None)

$ Example 1

- **MNIST & softmax**



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Softmax Train

- **load data**

  pass

- **Set Parameters**

  Personal habits： Storing parameters in a dictionary

```
Params['W'] = tf.Variable(tf.zeros([784,10]))

Params['b'] = tf.Variable(tf.zeros([10]))
```

Softmax Train

- **set Placeholder**

```
x = tf.placeholder(tf.float32, [None, 784])

y_ = tf.placeholder("float", [None,10])
```

- **Build graph**

```
y = tf.nn.softmax(tf.matmul(x, params['W']) + params['b'])

cross_entropy = -tf.reduce_sum(y_*tf.log(y))

train_step =

tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

Softmax Train

- **initialize variable op**

  Before Variables can be used within a session, they must be initialized using that session.

```
init = tf.initialize_all_variables()
```

- **Launch the graph in a session**

```
with tf.Session() as sess:
    sess.run(init)
    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    saver = tf.train.Saver(tf.all_variables())
    saver.save(sess, output_file_name)
```

Softmax Train

- **If we want to observer the loss:**

```
with tf.Session() as sess:
    sess.run(init)
    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        _,loss=sess.run([train_step, cross_entropy ], feed_dict={x: batch_xs,
                                                                 y_: batch_ys})
    saver = tf.train.Saver(tf.all_variables())
    saver.save(sess, output_file_name)
```

Softmax Train

- **Evaluation**
  - **Build graph**

```
y = tf.nn.softmax(tf.matmul(x, params['W']) + params['b']))

cross_entropy = -tf.reduce_sum(y_*tf.log(y))

train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

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

accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

Softmax Train

- **Launch the graph in a session**

```
with tf.Session() as sess:
    sess.run(init)
    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        _,loss=sess.run([train_step, cross_entropy], feed_dict={x: batch_xs,
                                                                 y_: batch_ys})
        if i%100==0:
            print sess.run(accuracy, feed_dict={x: mnist.test.images,
                                                y_: mnist.test.labels})
    saver = tf.train.Saver(tf.all_variables())
    saver.save(sess, output_file_name)
```

Softmax Test

- **test**
  - Load data
  - Set parameters
  - Set Placeholder
  - **Build graph**
  - **Load parameters**
  - **Launch the graph in a session**

Softmax Test

- **Build graph**

```
y = tf.nn.softmax(tf.matmul(x,params['W']) + params['b'])

class = tf.argmax(y,1)
```

- **Load parameters**

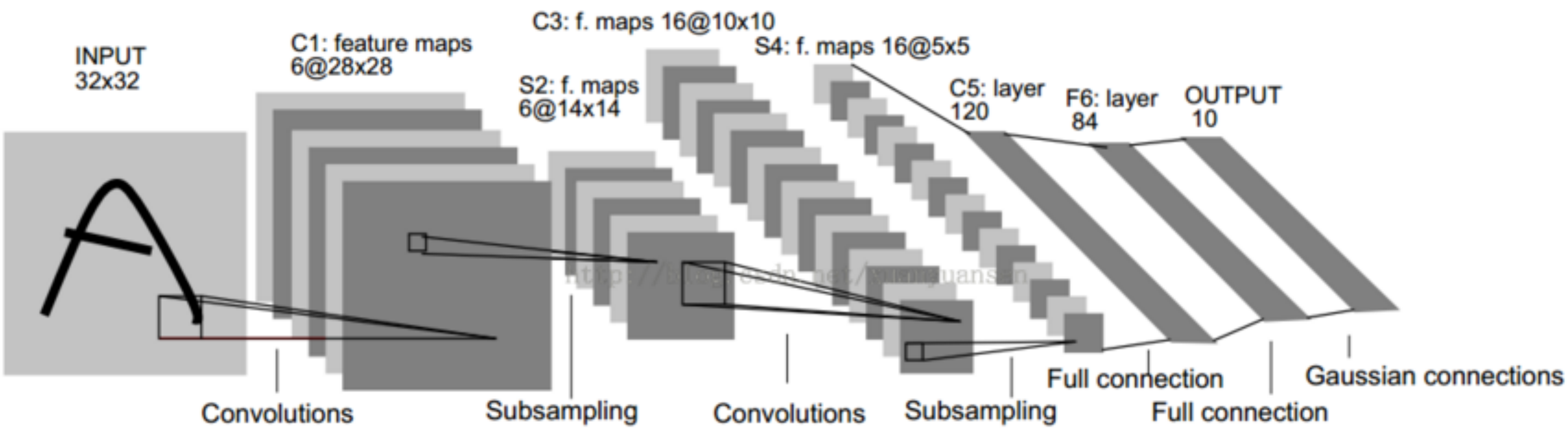
```
saver = tf.train.Saver(tf.all_variables())
```

- **Launch the graph in a session**

```
with tf.Session() as sess:
    saver.restore(sess, model_path)
    c = sess.run(class,feed_dict={x:test_x})
    print c
```

$ Example 2

- **MNIST & CNN**



http://blog.csdn.net/qiaofangjie/article/details/16826849

conv2d

- **conv2d(input, filter, strides, padding, use_cudnn_on_gpu=None, data_format=None, name=None)**

Computes a 2-D convolution given 4-D `input` and `filter` tensors.

Args:

input: A `Tensor`. Must be one of the following types: `float32`, `float64`.

filter: A `Tensor`. Must have the same type as `input`.

strides: A list of `ints`.1-D of length 4. The stride of the sliding window for each dimension of `input`. Must be in the same order as the dimension specified with format.

padding: A `string` from: `"SAME", "VALID"`.The type of padding algorithm to use.

use_cudnn_on_gpu: An optional `bool`. Defaults to `True`.

conv2d

data_format: An optional `string` from: `"NHWC", "NCHW"`. Defaults to `"NHWC"`. Specify the data format of the input and output data. With the default format "NHWC", the data is stored in the order of: [batch, in_height, in_width, in_channels]. Alternatively, the format could be "NCHW", the data storage order of: [batch, in_channels, in_height, in_width].

Returns:

A `Tensor`. Has the same type as `input`.

conv2d

Given an input tensor of shape `[batch, in_height, in_width, in_channels]` and a filter / kernel tensor of shape `[filter_height, filter_width, in_channels, out_channels]`, this op performs the following:

1. Flattens the filter to a 2-D matrix with shape `[filter_height * filter_width * in_channels, output_channels]`.
2. Extracts image patches from the input tensor to form a *virtual* tensor of shape `[batch, out_height, out_width, filter_height * filter_width * in_channels]`.
3. For each patch, right-multiplies the filter matrix and the image patch vector.

conv2d

In detail, with the default NHWC format,

output[b, i, j, k] =
sum_{di, dj, q} input[b, strides[1] * i + di, strides[2] * j + dj, q] *
filter[di, dj, q, k]

Must have `strides[0] = strides[3] = 1`. For the most common case of the same horizontal and vertices strides, `strides = [1, stride, stride, 1]`.

dropout

- **dropout(x, keep_prob, noise_shape=None, seed=None, name=None)**

Computes dropout.

With probability `keep_prob`, outputs the input element scaled up by `1 / keep_prob`, otherwise outputs `0`. The scaling is so that the expected sum is unchanged.

By default, each element is kept or dropped independently. If `noise_shape` is specified, it must be [broadcastable](http://docs.scipy.org/doc/numpy/user/basics.broadcasting.html) to the shape of `x`, and only dimensions with `noise_shape[i] == shape(x)[i]` will make independent decisions. For example, if `shape(x) = [k, l, m, n]` and `noise_shape = [k, 1, 1, n]`, each batch and channel component will be kept independently and each row and column will be kept or not kept together.

dropout

Args:

x: A tensor.

keep_prob: A scalar `Tensor` with the same type as x. The probability that each element is kept.

noise_shape: A 1-D `Tensor` of type `int32`, representing the shape for randomly generated keep/drop flags.

seed: A Python integer. Used to create random seeds. See [`set_random_seed`](../../api_docs/python/constant_op.md#set_random_seed) for behavior.

Returns:

A Tensor of the same shape of `x`.

Raises:

ValueError: If `keep_prob` is not in `(0, 1]`.

Max_pool

- **max_pool(value, ksize, strides, padding, data_format='NHWC', name=None)**

Performs the max pooling on the input.

Args:

value: A 4-D `Tensor` with shape `[batch, height, width, channels]` and type `tf.float32`.

ksize: A list of ints that has length >= 4. The size of the window for each dimension of the input tensor.

strides: A list of ints that has length >= 4. The stride of the sliding window for each dimension of the input tensor.

padding: A string, either `'VALID'` or `'SAME'`. The padding algorithm.

data_format: A string. 'NHWC' and 'NCHW' are supported.

Returns:

A `Tensor` with type `tf.float32`. The max pooled output tensor.

CNN Train

- **Set Parameters**

```
def weight_variable(shape):
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)

def bias_variable(shape):
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)
```

```
params[‘W_conv1’] = weight_variable([5, 5, 1, 32])
params[‘b_conv1’] = bias_variable([32])

params[‘W_conv2’] = weight_variable([5, 5, 32, 64])
params[‘b_conv2’] = bias_variable([64])

params[‘W_fc1’] = weight_variable([7 * 7 * 64, 1024])
params[‘ b_fc1’] = bias_variable([1024])

params[‘W_fc2’] = weight_variable([1024, 10])
params[‘b_fc2’] = bias_variable([10])
```

CNN Train

- **set placeholder**

```
x = tf.placeholder("float", shape=[None, 784])
x_image = tf.reshape(x, [-1,28,28,1])

y_ = tf.placeholder("float", shape=[None, 10])

keep_prob = tf.placeholder("float")
```

CNN Train

- **build graph**

```
def conv2d(x, W):
     return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

h_conv1 = tf.nn.relu(conv2d(x_image, params['W_conv1'] ) + params['b_conv1'] )
h_pool1 = max_pool_2x2(h_conv1)

h_conv2 = tf.nn.relu(conv2d(h_pool1, params['W_conv2'] ) + params['b_conv2'] )
h_pool2 = max_pool_2x2(h_conv2)

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])

h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat,params['W_fc1'])  + params['b_fc1'])

h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, params['W_fc2']) ) + params['b_fc2']) )
```

CNN Train

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
saver = tf.train.Saver(tf.all_variables())
```

tf.cast： convert the Boolean value into float

$ Example 3

Based DNN TTS

$ Example 4

Based-RNN TTS