深度学习工具实战

龚经经

复旦大学

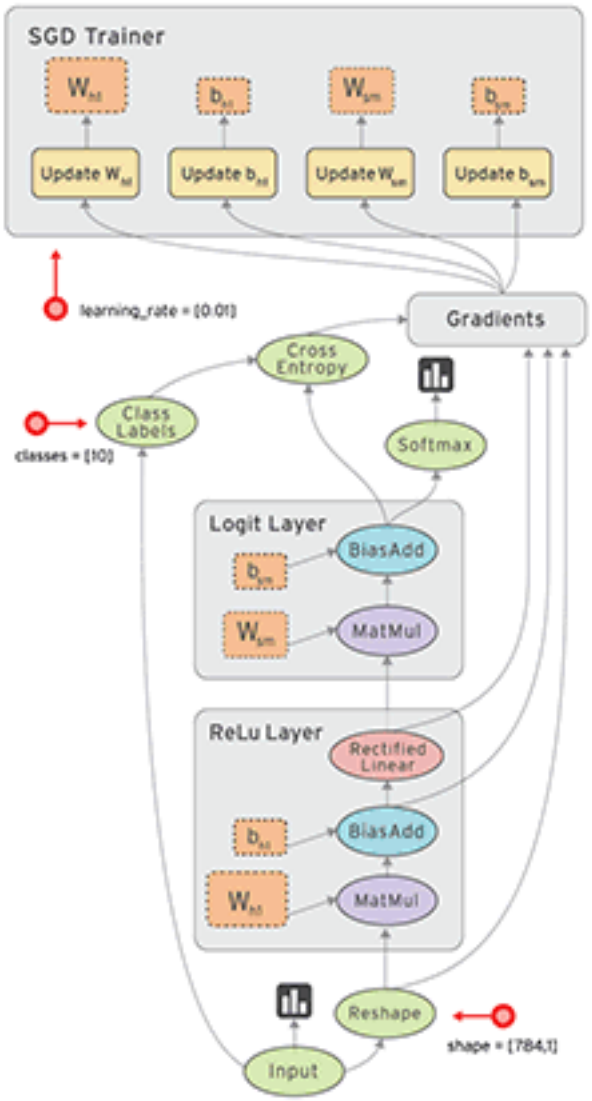
jjgong15@fudan.edu.cn

关于TensorFlow

- Tensorflow 是一个开源的基于数据流图的数学计算库
- google brain开发用来做机器学习深度学习研究的工具
- 多平台支持，服务器，个人电脑，移动设备
- 支持多卡，分布式计算
- 灵活，具有通用性，可以用在其它领域
- 当前支持python, java以及c++接口
- **可以自动求导**
- **可以方便的利用GPU**

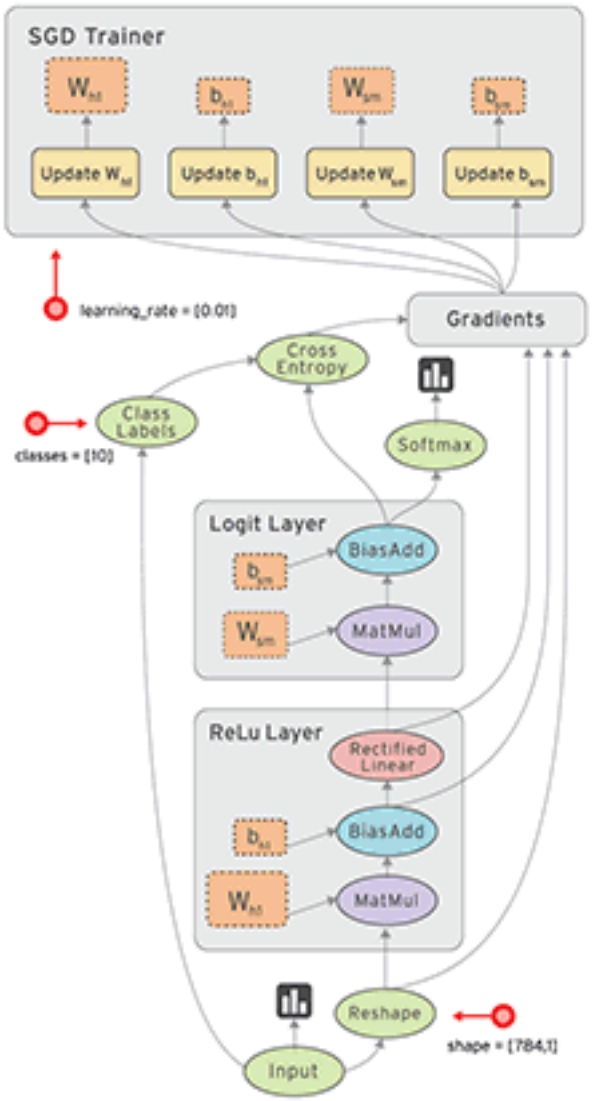
Data Flow Graph(数据流图)

- 数据流图是一个用来描述数学计算的有向图(有向无环图)
- 节点代表操作(Operation)
- 连接节点的边代表Operation的输出或者称作tensor
- 每一个节点的激活需要所有前驱节点激活



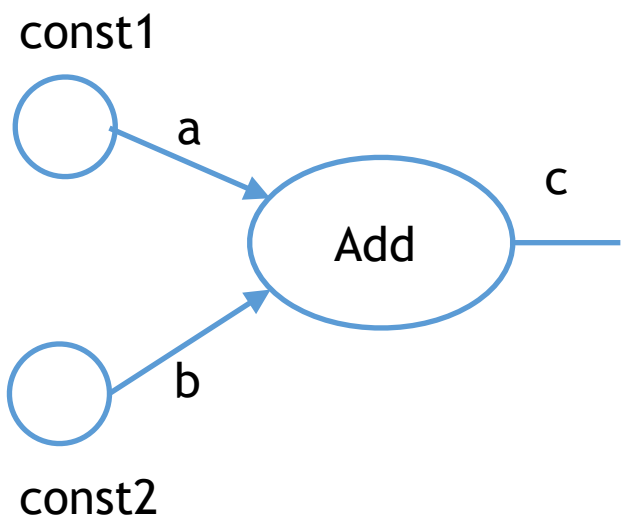
Data Flow Graph(数据流图)

- 数据流图是一个用来描述数学计算的有向图(有向无环图)
- 节点代表操作(Operation)
- 连接节点的边代表Operation的输出或者称作tensor
- 每一个节点的激活需要所有前驱节点激活



Tensorflow基础

写tensorflow代码的流程:建图+执行

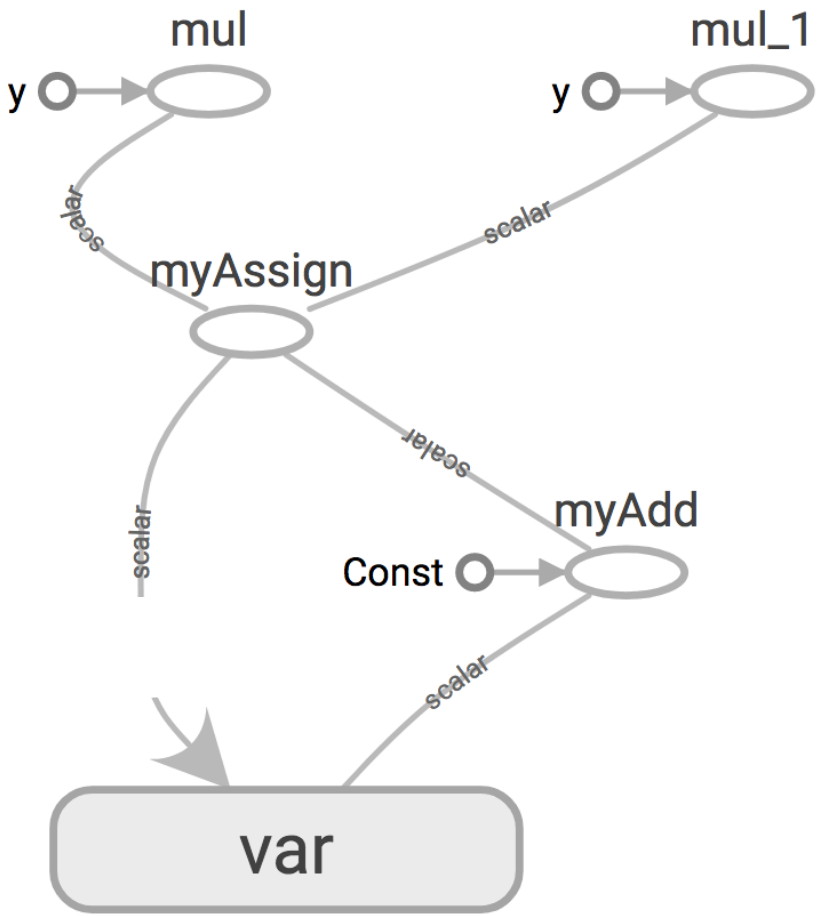



```
import tensorflow as tf
import numpy as np

a = tf.constant(1., name='const1')
b = tf.constant(2., name='const2')
c = tf.add(a, b)
with tf.Session() as sess:
    print sess.run(c)
    print c.eval()
```

```
3.0
3.0
```

终端节点

- tf.Variable。变量节点
  - 用来存在图执行过程中需要更新的量，
  - 在神经网络中用来存储权值
- tf.constant。常量节点
  - 在建图时值已经确定，所以要传一个 python 值，而不能传一个tensor
- tf.placeholder。占位节点
  - 在运行时要给它喂/feed 一个值
- tf.zeros, tf.ones, tf.zeros_like, tf.ones_like, tf.random , …



Example: random number generator

```
import numpy as np
aa = np.random.rand(1)

for i in range(5):
    print aa
```

```
[ 0.12619646]
[ 0.12619646]
[ 0.12619646]
[ 0.12619646]
[ 0.12619646]
```

```
import tensorflow as tf
import numpy as np

a = tf.random_normal([1], name='random')
with tf.Session() as sess:
    for i in range(5):
        print sess.run(a)
```

Example: random number generator

```
import numpy as np
aa = np.random.rand(1)

for i in range(5):
    print aa
```

```
[ 0.12619646]
[ 0.12619646]
[ 0.12619646]
[ 0.12619646]
[ 0.12619646]
```

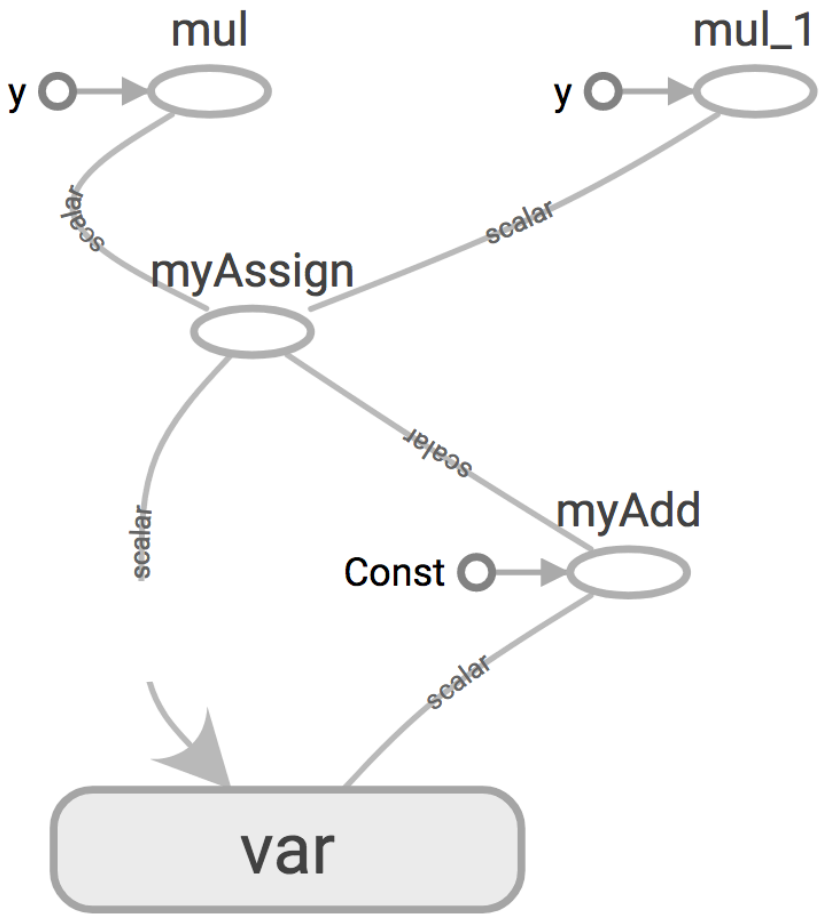
```
import tensorflow as tf
import numpy as np

a = tf.random_normal([1], name='random')
with tf.Session() as sess:
    for i in range(5):
        print sess.run(a)
```

```
[ 0.38659823]
[-0.6114915]
[ 0.22402158]
[ 0.48937175]
[-0.63917536]
```

feed& fetch

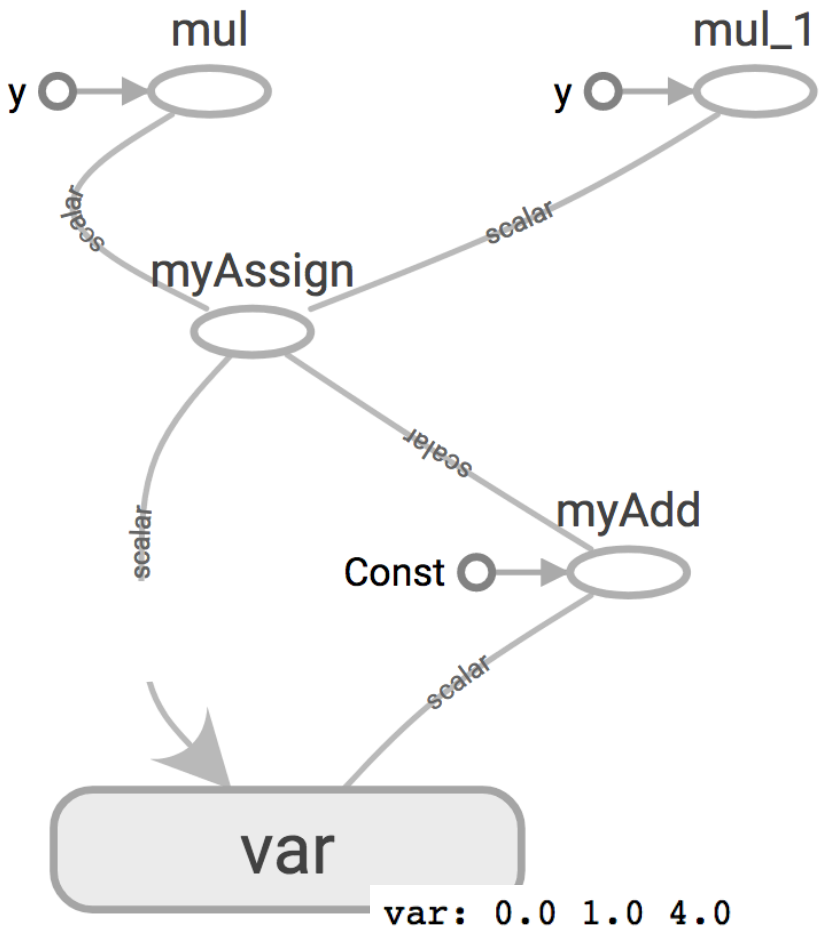
- feed 和fetch是python与tensorflow数据交流的途径
  - feed 将数据喂进tensorflow实例图（喂给placeholder节点）
  - fetch tensorflow实例图中取数据。

Example: assign, fetch




```python
#Assign example
import tensorflow as tf
import numpy as np

var = tf.Variable(0., name='var')
const = tf.constant(1.)
add_op = tf.add(var, const, name='myAdd') # tmp = var + const

#return a tensor, and have a side effect, that is assign add_op to var
assign_op = tf.assign(var, add_op, name='myAssign')

out1 = assign_op*1
out2 = assign_op*2

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(3):
        print "var:", sess.run(var), sess.run(out1), sess.run(out2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(3):
        print "var:", sess.run(var), sess.run([out1, out2])
```

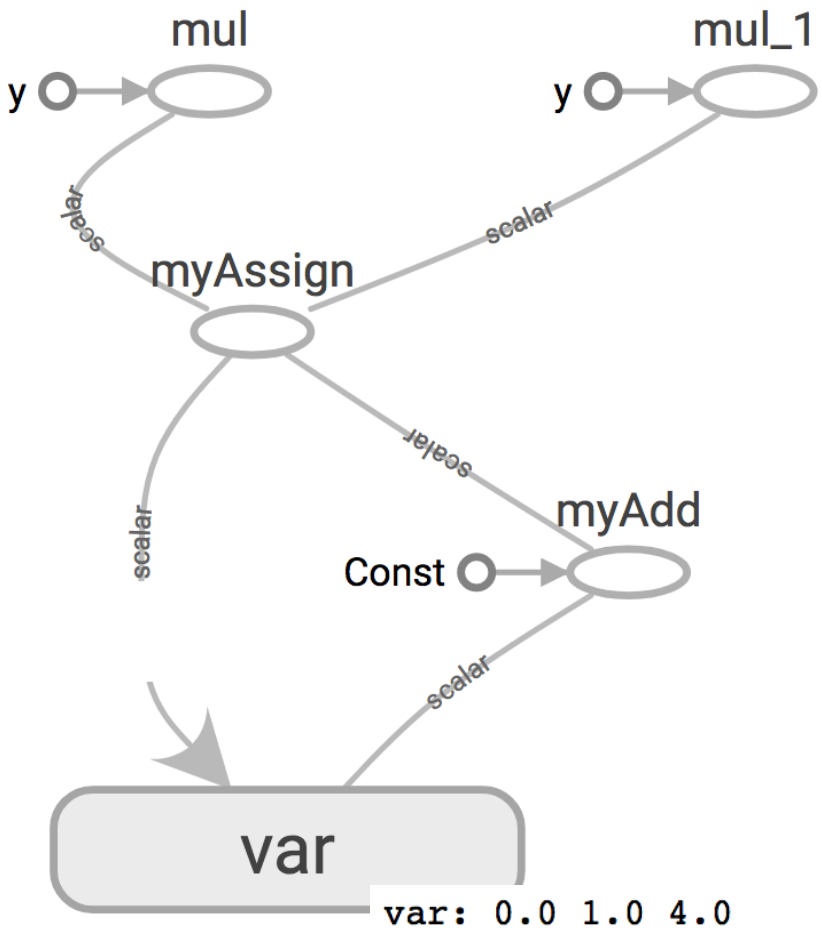
Example: assign, fetch




```
var: 0.0 1.0 4.0
var: 2.0 3.0 8.0
var: 4.0 5.0 12.0
var: 0.0 [1.0, 2.0]
var: 1.0 [2.0, 4.0]
var: 2.0 [3.0, 6.0]
```

```python
#Assign example
import tensorflow as tf
import numpy as np

var = tf.Variable(0., name='var')
const = tf.constant(1.)
add_op = tf.add(var, const, name='myAdd') # tmp = var + const

#return a tensor, and have a side effect, that is assign add_op to var
assign_op = tf.assign(var, add_op, name='myAssign')

out1 = assign_op*1
out2 = assign_op*2

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(3):
        print "var:", sess.run(var), sess.run(out1), sess.run(out2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(3):
        print "var:", sess.run(var), sess.run([out1, out2])
```

Example: assign, fetch




```
var: 0.0 1.0 4.0
var: 2.0 3.0 8.0
var: 4.0 5.0 12.0
var: 0.0 [1.0, 2.0]
var: 1.0 [2.0, 4.0]
var: 2.0 [3.0, 6.0]
```

```python
#Assign example
import tensorflow as tf
import numpy as np

var = tf.Variable(0., name='var')
const = tf.constant(1.)
add_op = tf.add(var, const, name='myAdd') # tmp = var + const

#return a tensor, and have a side effect, that is assign add_op to var
assign_op = tf.assign(var, add_op, name='myAssign')

out1 = assign_op*1
out2 = assign_op*2

out1 = tf.multiply(assign_op, 1)
out2 = tf.multiply(assign_op, 2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(3):
        print "var:", sess.run(var), sess.run(out1), sess.run(out2)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(3):
        print "var:", sess.run(var), sess.run([out1, out2])
```

Numpy to tensorflow Dictionary

Numpy	Tensorflow
a = np.zeros([2, 2]); b = np.ones([2, 2])	a = tf.zeros([2, 2); b = tf.ones([2, 2])
np.rum(b, axis=1)	tf.reduce_sum(a, axis=1)
a.shape	tf.shape(a)
np.reshape(a, [1, 4])	tf.reshape(a, [1, 4])
b*5+1	b*5+1
np.dot(a, b)	tf.matmul(a, b)
a[0, 0], a[:, 0], a[0, :]	a[0, 0], a[:, 0], a[0, :]

如何快捷地访问一个变量节点?

- 对于一个复杂模型来说，可能会有上百个变量节点，如何访问其中一个变量节点?

- 利用Scope来对变量节点进行分组和访问
  - 每个Scope可以对应到一个神经层或子模块

- tf.variable_scope() 提供一种简单的命名空间机制
  - 在一个scope下面可以通过tf.get_variable() 在当前scope下 创建 / 复用 一个Variable

Variable Scope

- 在tensorflow 中所有的变量都是全局变量，Variable scope起到了划分命名空间的作用，给一个Variable名字加一个前缀

```
import tensorflow as tf

with tf.variable_scope('foo'):
    with tf.variable_scope('dummy') as sp:
        v = tf.get_variable('v', shape=[1, 2])
    v2 = tf.get_variable('v2', shape=[1, 1])

print v.name
print v2.name
```

```
foo/dummy/v:0
foo/v2:0
```

Variable Scope & Variable reuse

- 可以使用tf.get_variable() 创建一个Variable节点
- 当要引用一个variable的时候调用tf.get_variable() 可以得到一个Variable节点的引用（把scope中的reuse置True）

```
import tensorflow as tf
with tf.variable_scope("foo"):
    aaa = tf.get_variable("aaa", [1])
    bbb = tf.get_variable("bbb", [1])
with tf.variable_scope("foo", reuse=True):
    ccc = tf.get_variable("aaa")

print aaa.name
print bbb.name
print ccc.name
```

```
foo/aaa:0
foo/bbb:0
foo/aaa:0
```

Variable Scope & Variable reuse

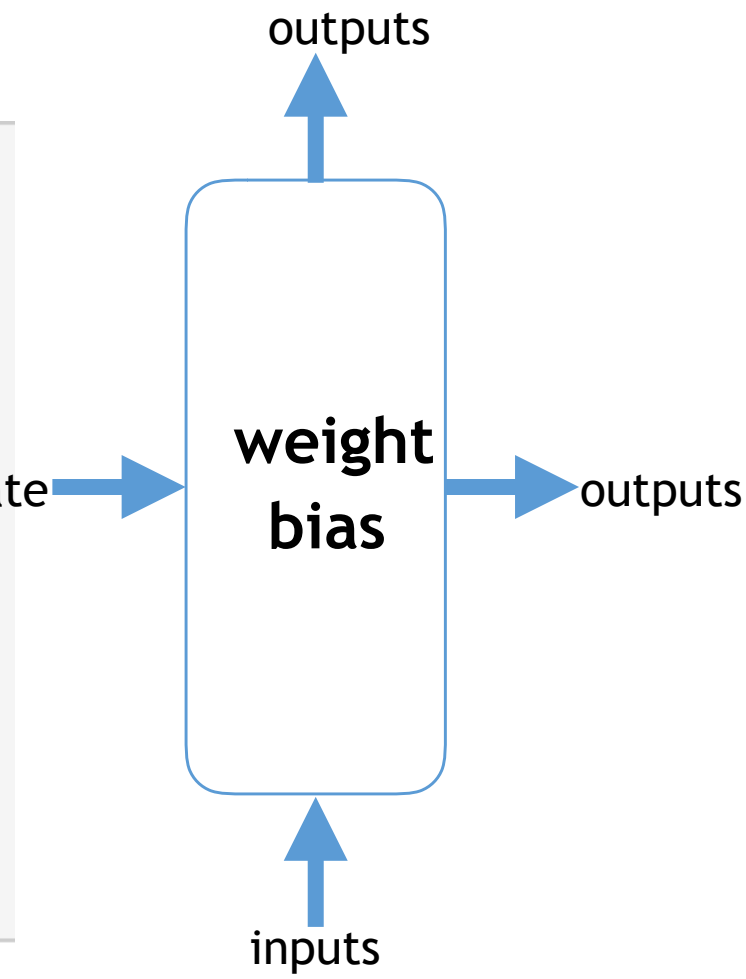
- 另外一种Variable引用方式

```
import tensorflow as tf
with tf.variable_scope("foo") as scope:
    aaa = tf.get_variable("aaa", [1])
    bbb = tf.get_variable("bbb", [1])
    scope.reuse_variables()
    ccc = tf.get_variable("aaa")
print aaa.name
print bbb.name
print ccc.name
```

```
foo/aaa:0
foo/bbb:0
foo/aaa:0
```

Variable Scope & Variable reuse

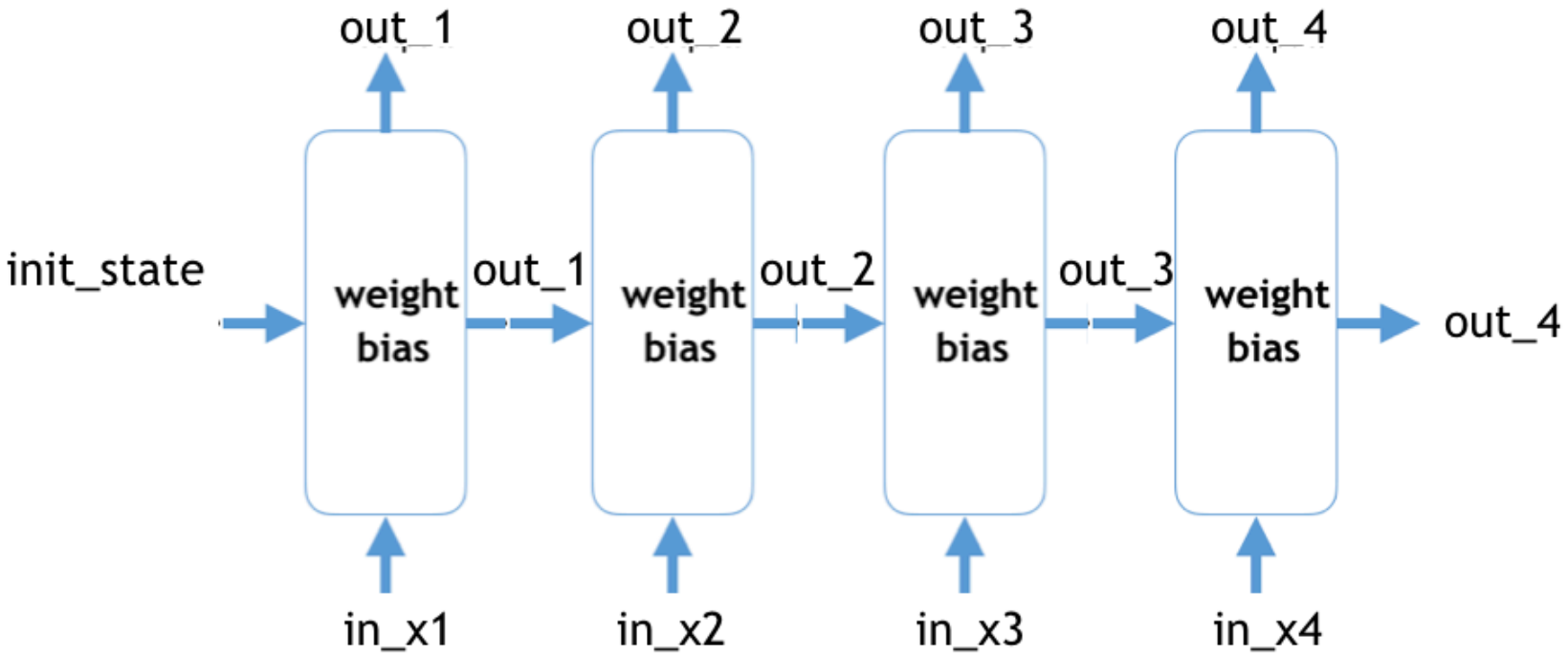
```
def rnn(inputs, state, hidden_size):
    in_x = tf.concat([inputs, state], axis=1)
    W_shape = [int(in_x.get_shape()[1]), hidden_size]
    b_shape = [1, hidden_size]

    W = tf.get_variable(shape=W_shape, name='weight')
    b = tf.get_variable(shape=b_shape, name='bias')

    out_linear = tf.nn.bias_add(tf.matmul(in_x, W), b)

    output = tf.nn.tanh(out_linear)
    return output
```



Variable Scope & Variable reuse

```
out_1 = rnn(in_x1, init_state, 64)
out_2 = rnn(in_x2, out_1, 64)
out_3 = rnn(in_x3, out_2, 64)
out_4 = rnn(in_x4, out_3, 64)
```

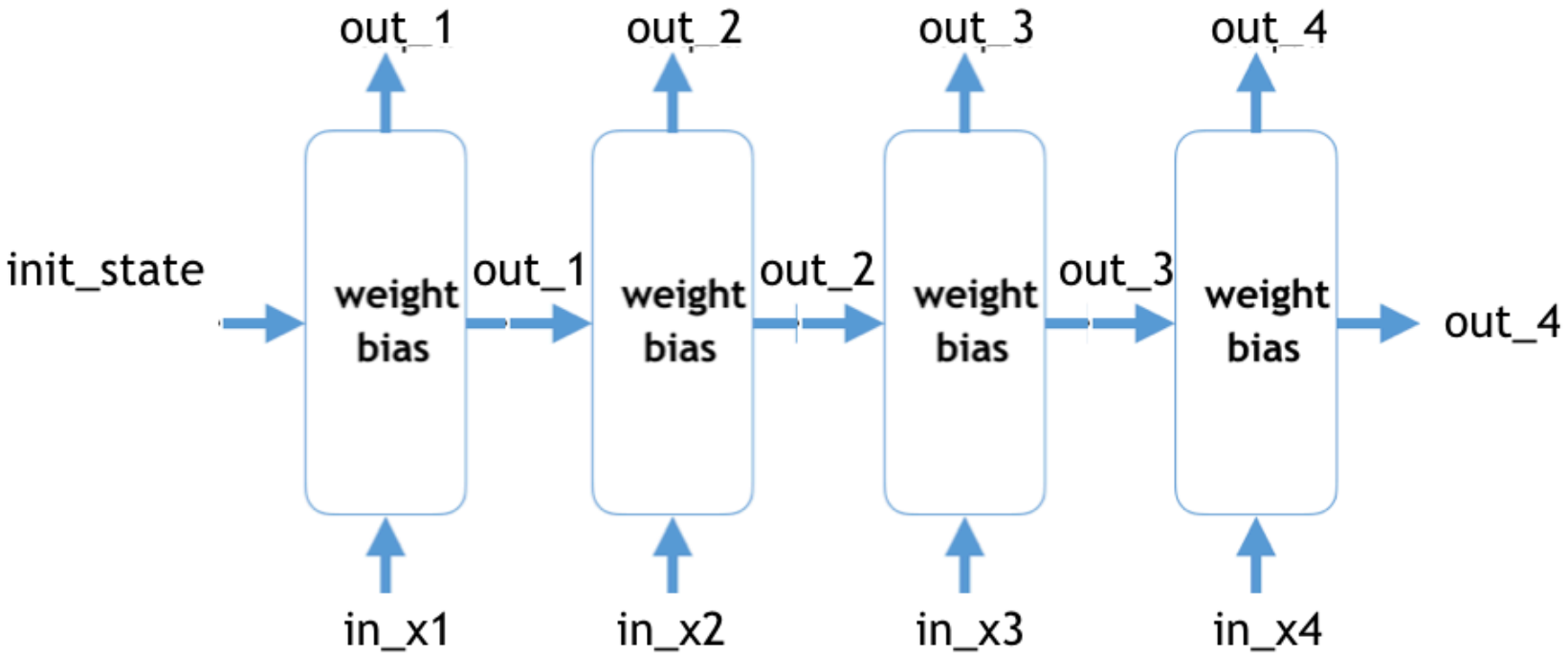
```
with tf.variable_scope('rnn_scope') as scope:
    out_1 = rnn(in_x1, init_state, 64)
    scope.reuse_variables()
    out_2 = rnn(in_x2, out_1, 64)
    out_3 = rnn(in_x3, out_2, 64)
    out_4 = rnn(in_x4, out_3, 64)
```



Variable Scope & Variable reuse

```
out_1 = rnn(in_x1, init_state, 64)
out_2 = rnn(in_x2, out_1, 64)
out_3 = rnn(in_x3, out_2, 64)
out_4 = rnn(in_x4, out_3, 64)
```

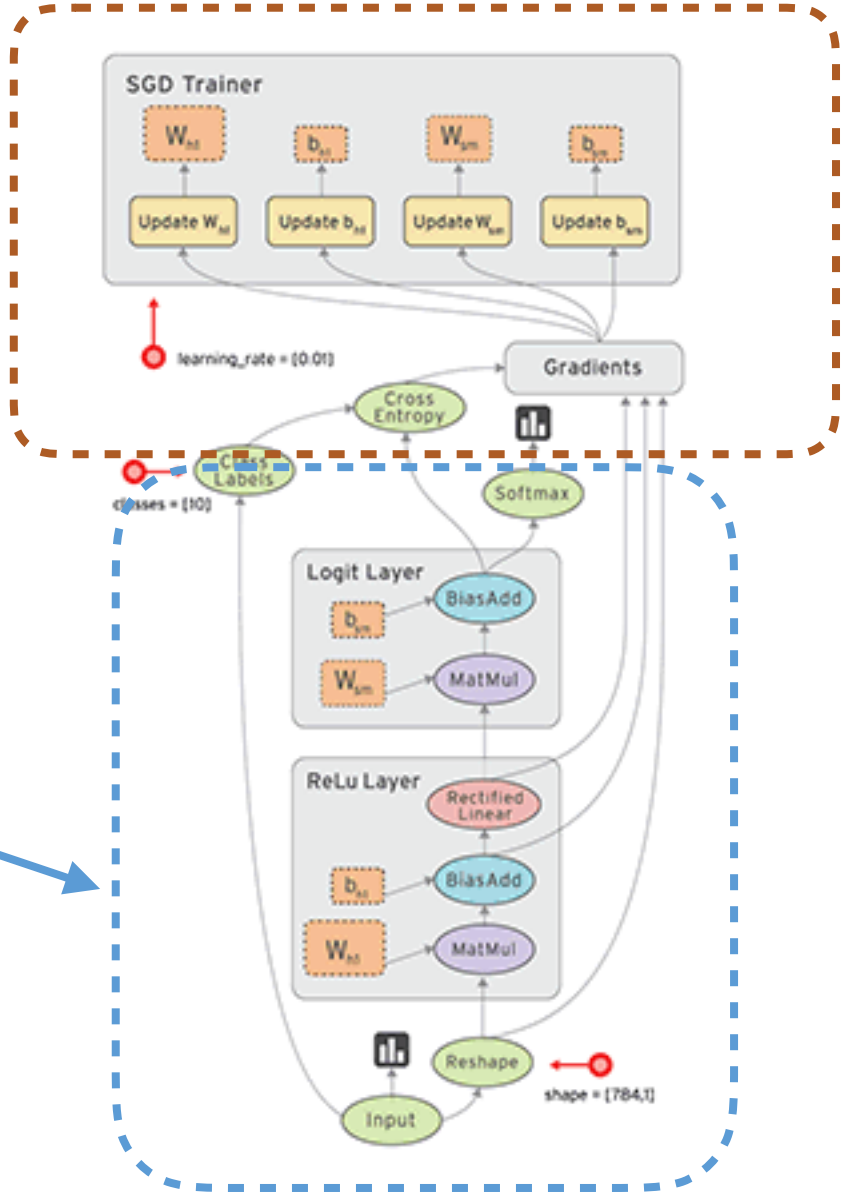
```
with tf.variable_scope('rnn_scope') as scope:
    out_1 = rnn(in_x1, init_state, 64)
    scope.reuse_variables()
    out_2 = rnn(in_x2, out_1, 64)
    out_3 = rnn(in_x3, out_2, 64)
    out_4 = rnn(in_x4, out_3, 64)
```



Tensorflow机器学习实践

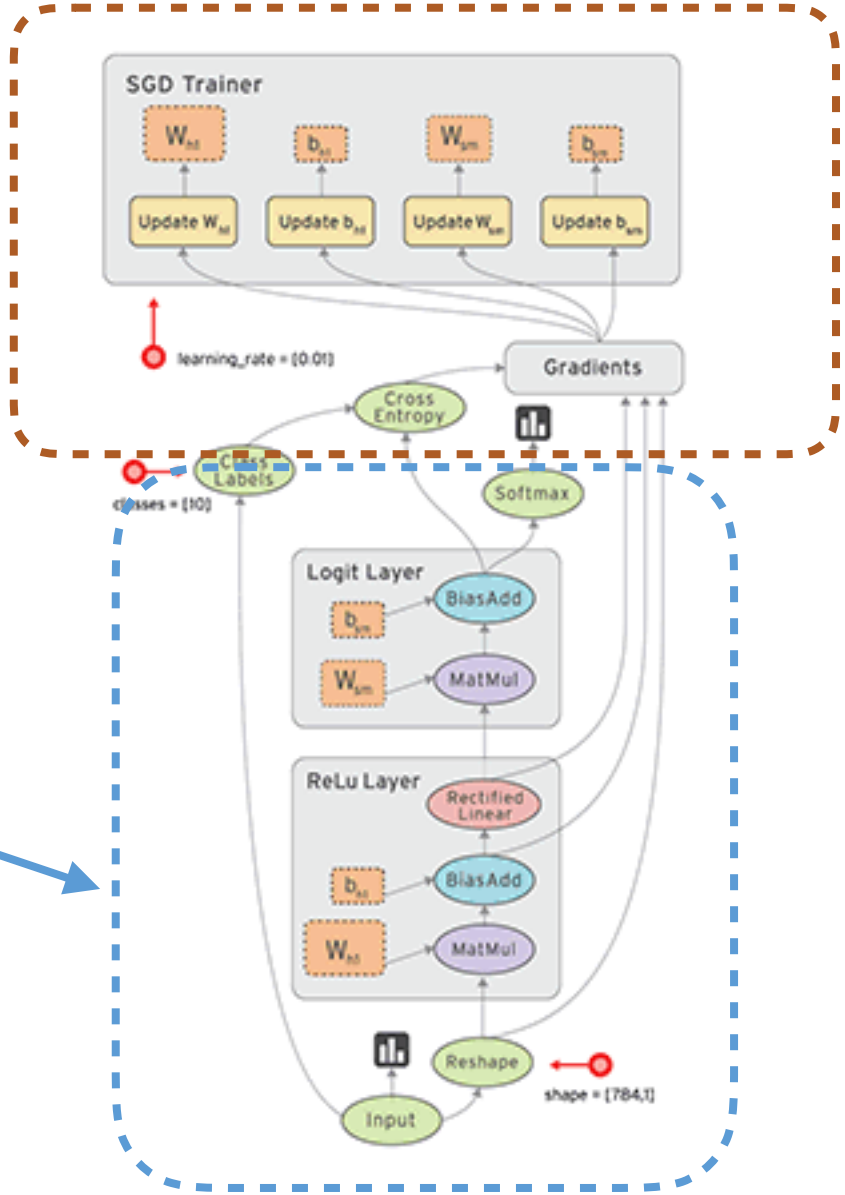
机器学习

- 机器学习
  - 模型
  - 优化
    - 损失函数
    - 梯度
    - 优化（梯度下降）



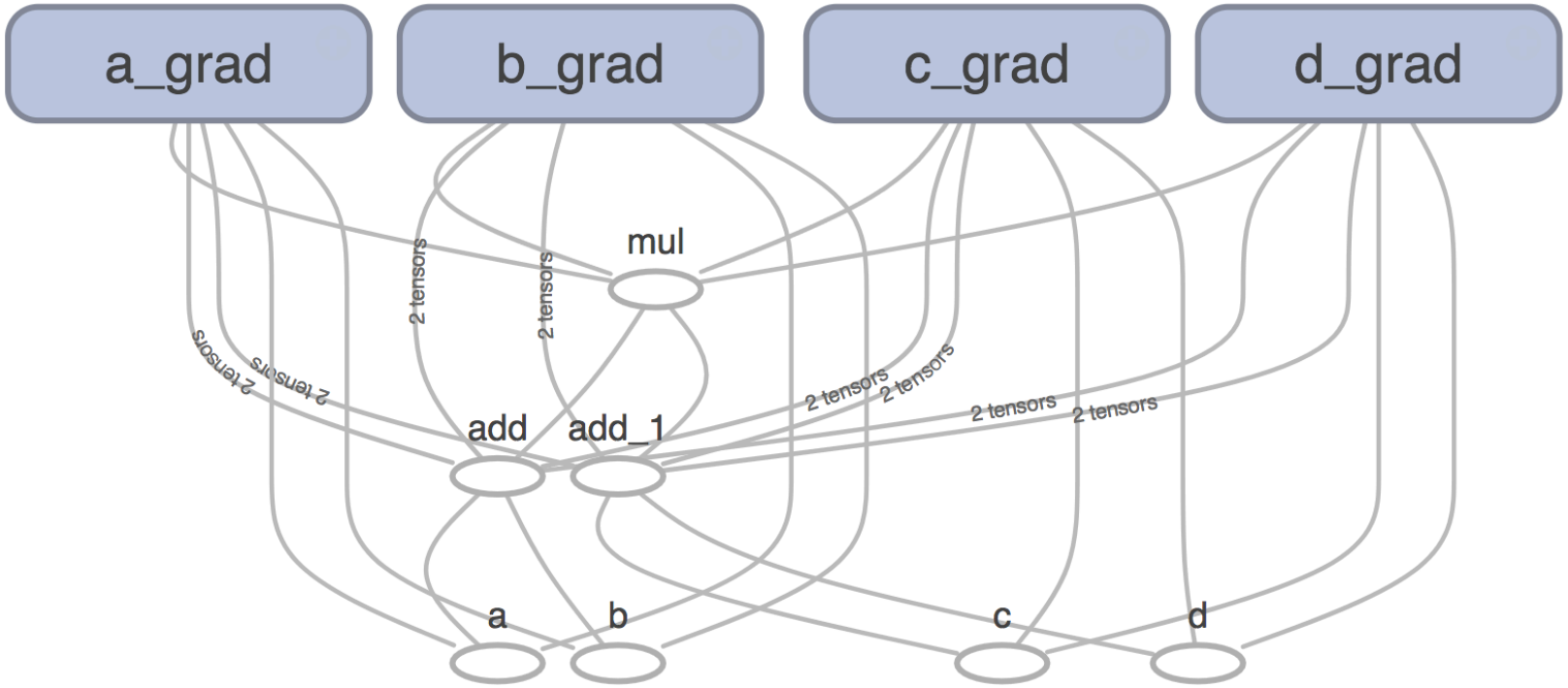
机器学习

- 机器学习
  - 模型
  - 优化
    - 损失函数
    - 梯度
    - 优化（梯度下降）



Gradient of $e = (a + b) * (c + d)$ in tensorflow

```
import tensorflow as tf

a = tf.placeholder(shape=None, dtype=tf.float32, name='a')
b = tf.placeholder(shape=None, dtype=tf.float32, name='b')
c = tf.placeholder(shape=None, dtype=tf.float32, name='c')
d = tf.placeholder(shape=None, dtype=tf.float32, name='d')

'''inference'''
e = (a+b)*(c+d)

'''caculate gradient'''
a_grad, b_grad, c_grad, d_grad =(
    tf.gradients(e, a, name='a_grad'),
    tf.gradients(e, b, name='b_grad'),
    tf.gradients(e, c, name='c_grad'),
    tf.gradients(e, d, name='d_grad')
)

with tf.Session() as sess:
    (grad_a, grad_b, grad_c, grad_d
    ) = sess.run([a_grad, b_grad,
                  c_grad, d_grad],
                 feed_dict={a:1., b:2.,
                            c:6., d:4.})
    print grad_a, grad_b
    print grad_c, grad_d
```

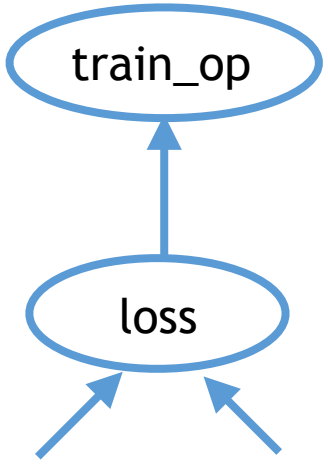



```
[10.0] [10.0]
[3.0] [3.0]
```

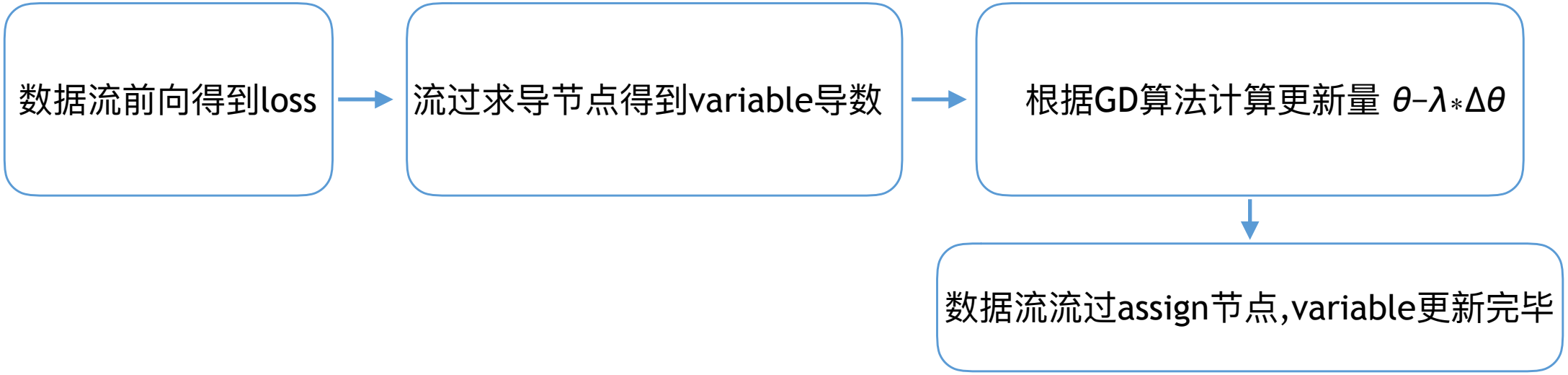
Optimizer

- tf.train.GradientDescentOptimizer

```
opt = tf.train.GradientDescentOptimizer(0.01)
train_op = opt.minimize(loss)
```

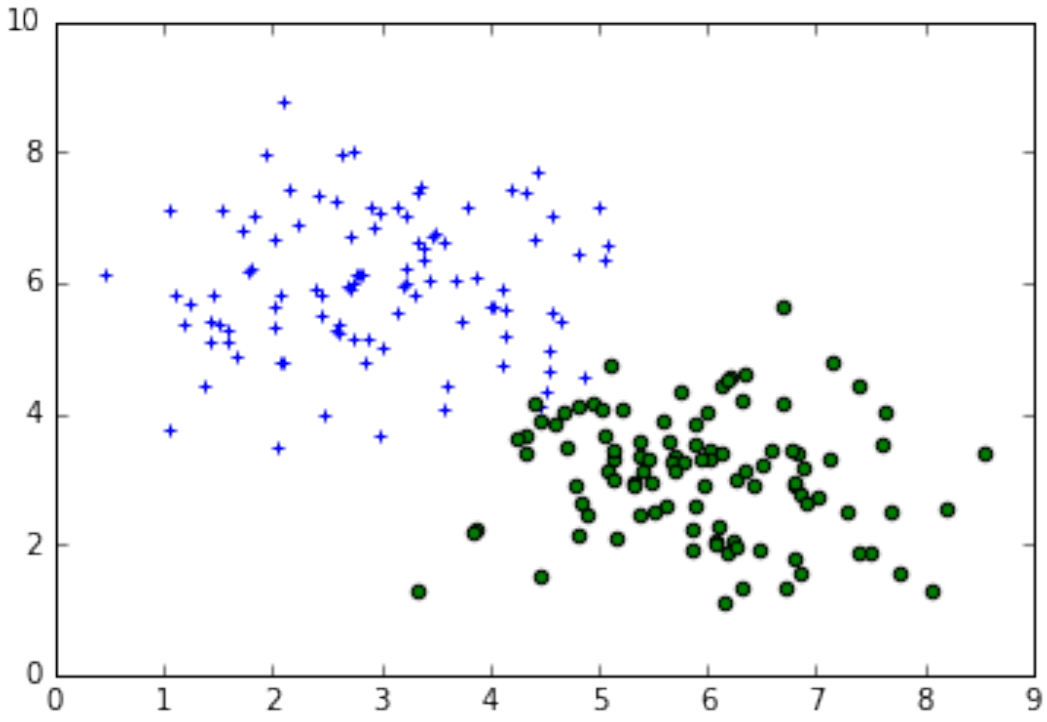


- 当对train_op进行fetch时：



Example: Logistic regression

- 数据
  - 两组数据（分别在图中用'+' 以及'o'表示）分别采样于两个二维高斯分布 $(x, y) \sim N(3, 6, 1, 1, 0)$ 以及 $(x, y) \sim N(6, 3, 1, 1, 0)$
- 目标
  - 将这两组采自不同分布的点用一条直线分开



Logistic regression : 建图

```
################Graph building stage################
# Define placeholders for input
X = tf.placeholder(tf.float32, shape=(None, 2))
y = tf.placeholder(tf.float32, shape=(None, 1))

# Define variables to be learned
with tf.variable_scope("Logistic_regression"):
    W = tf.Variable(np.random.rand(2, 1), 'weight', dtype=tf.float32)
    b = tf.Variable(np.random.rand(1, 1), 'bias', dtype=tf.float32)

    logits = tf.matmul(X, W) + b
    pred = tf.sigmoid(logits)
    loss = tf.nn.sigmoid_cross_entropy_with_logits(logits, y)
    loss = tf.reduce_mean(loss)

opt = tf.train.AdamOptimizer(0.01)
train_op = opt.minimize(loss)
```

Logistic regression : 建图

```
################Graph building stage################
# Define placeholders for input
X = tf.placeholder(tf.float32, shape=(None, 2))
y = tf.placeholder(tf.float32, shape=(None, 1))

# Define variables to be learned
with tf.variable_scope("Logistic_regression"):
    W = tf.Variable(np.random.rand(2, 1), 'weight', dtype=tf.float32)
    b = tf.Variable(np.random.rand(1, 1), 'bias', dtype=tf.float32)

    logits = tf.matmul(X, W) + b
    pred = tf.sigmoid(logits)
    loss = tf.nn.sigmoid_cross_entropy_with_logits(logits, y)
    loss = tf.reduce_mean(loss)

opt = tf.train.AdamOptimizer(0.01)
train_op = opt.minimize(loss)
```

定义占位节点，数据入口

Logistic regression : 建图

```
################Graph building stage################
# Define placeholders for input
X = tf.placeholder(tf.float32, shape=(None, 2))
y = tf.placeholder(tf.float32, shape=(None, 1))

# Define variables to be learned
with tf.variable_scope("Logistic_regression"):
    W = tf.Variable(np.random.rand(2, 1), 'weight', dtype=tf.float32)
    b = tf.Variable(np.random.rand(1, 1), 'bias', dtype=tf.float32)

    logits = tf.matmul(X, W) + b
    pred = tf.sigmoid(logits)
    loss = tf.nn.sigmoid_cross_entropy_with_logits(logits, y)
    loss = tf.reduce_mean(loss)

opt = tf.train.AdamOptimizer(0.01)
train_op = opt.minimize(loss)
```

定义占位节点，数据入口

建立参数节点

Logistic regression : 建图

```
################Graph building stage################
# Define placeholders for input
X = tf.placeholder(tf.float32, shape=(None, 2))
y = tf.placeholder(tf.float32, shape=(None, 1))

# Define variables to be learned
with tf.variable_scope("Logistic_regression"):
    W = tf.Variable(np.random.rand(2, 1), 'weight', dtype=tf.float32)
    b = tf.Variable(np.random.rand(1, 1), 'bias', dtype=tf.float32)

    logits = tf.matmul(X, W) + b
    pred = tf.sigmoid(logits)
    loss = tf.nn.sigmoid_cross_entropy_with_logits(logits, y)
    loss = tf.reduce_mean(loss)

opt = tf.train.AdamOptimizer(0.01)
train_op = opt.minimize(loss)
```

定义占位节点，数据入口

建立参数节点

构建loss

Logistic regression : 建图

```
################Graph building stage################
# Define placeholders for input
X = tf.placeholder(tf.float32, shape=(None, 2))
y = tf.placeholder(tf.float32, shape=(None, 1))

# Define variables to be learned
with tf.variable_scope("Logistic_regression"):
    W = tf.Variable(np.random.rand(2, 1), 'weight', dtype=tf.float32)
    b = tf.Variable(np.random.rand(1, 1), 'bias', dtype=tf.float32)

    logits = tf.matmul(X, W) + b
    pred = tf.sigmoid(logits)
    loss = tf.nn.sigmoid_cross_entropy_with_logits(logits, y)
    loss = tf.reduce_mean(loss)

opt = tf.train.AdamOptimizer(0.01)
train_op = opt.minimize(loss)
```

定义占位节点，数据入口

建立参数节点

构建loss

建立优化节点，每次数据流流过该节点都会更新一次参数，使得loss变小

Logistic regression：实例化图以及运行图

```
with tf.Session() as sess:     ##initialize a graph
    sess.run(tf.initialize_all_variables())
    for _ in xrange(500):
        _, loss_val = sess.run([train_op, loss],
                               feed_dict={X: data_set[:, :2],
                                          y: data_set[:, 2].reshape((-1, 1))})
```

- 实例化并且feed&fetch 实例图
  - 实例化: 将一幅静态的图实例化成一个可运行实体（类似于创建一个进程）
  - feed: 将外部数据喂给占位节点（placeholder）
  - fetch: 例子中train_op 以及 loss都是fetch对象，fetch train_op 目的是让loss得到优化。fetch loss是为了得到当前的loss值

Logistic regression：可视化结果

Logistic regression：可视化结果

Logistic regression：可视化结果

Tensorflow 深度学习实践

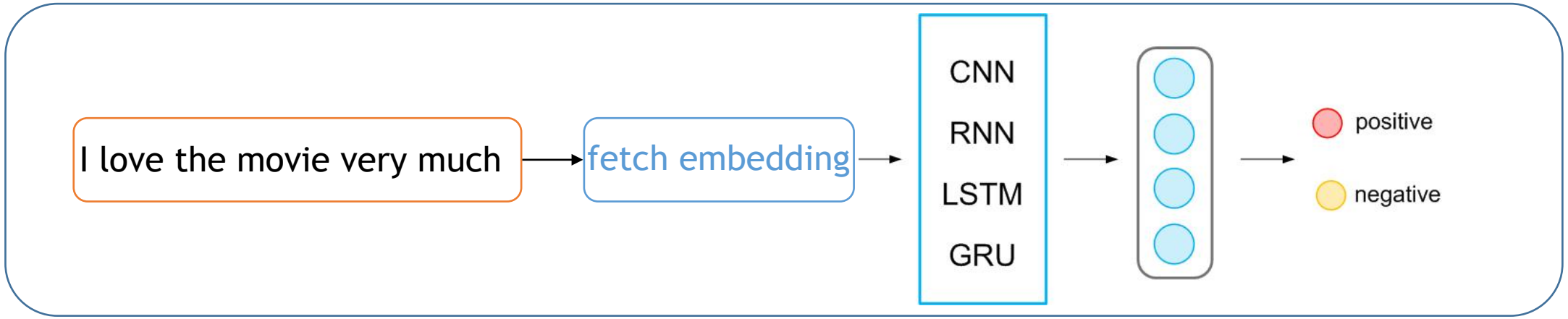
Example：影评的情感分类

- 任务描述
  - 输入：一个句子
  - 输出：分类结果（情感的正负）

正面：I love this movie.	[0, 1]
负面：The director could do better than this.	[1, 0]

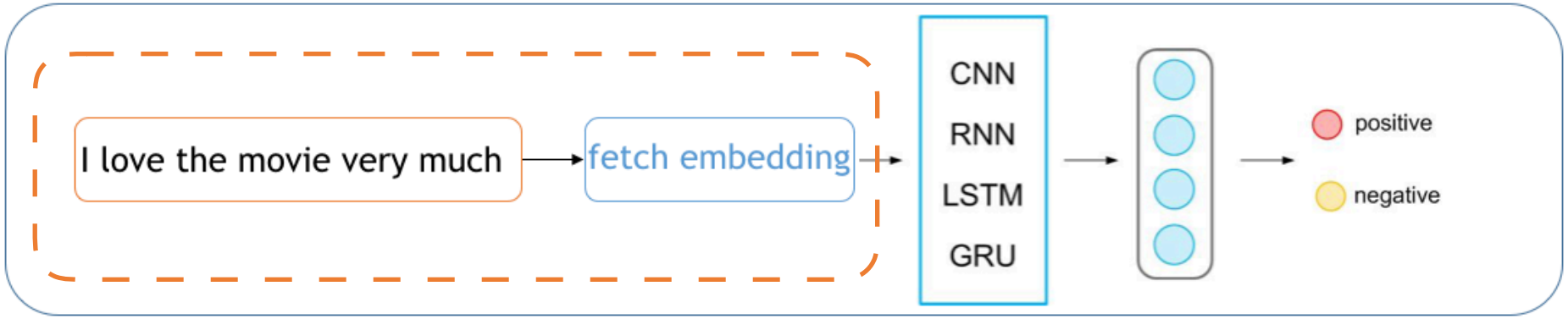
- 数据集: Movie Review data from Rotten Tomatoes
  - 10,662 句子
  - 正负各半

一般解决思路



1. 将文本先映射到编号序列，并且按照编号查找embedding
2. 得到的embedding 向量序列 输入到一个序列编码器(CNN，RNN，LSTM, GRU, etc.) 得到一个对文本的向量表示
3. 将文本的向量表示映射到2类分类空间(全联接+ softmax)

输入向量化



I love the movie very much

```
vocab = {'I':123, 'love': 234, 'the':345, 'movie':456,...}
```

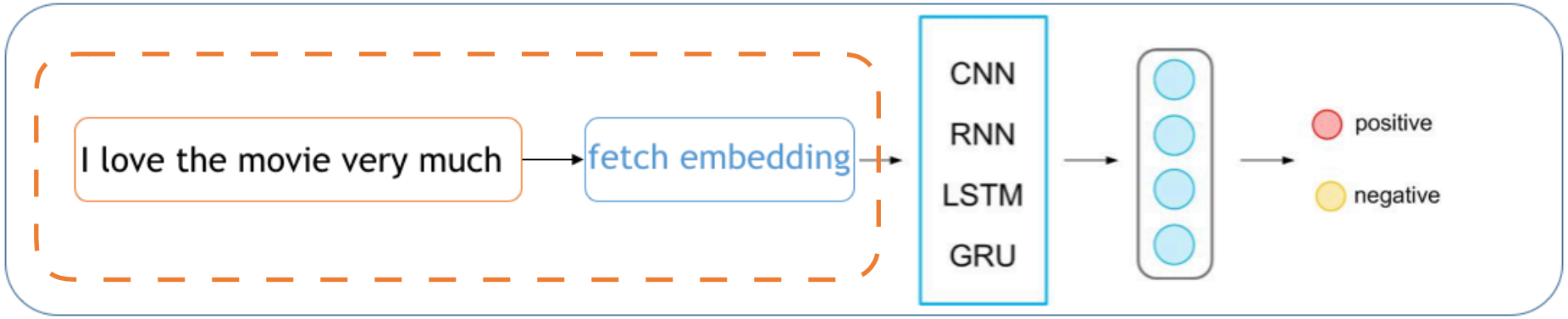
[vocab[o] for o in text]

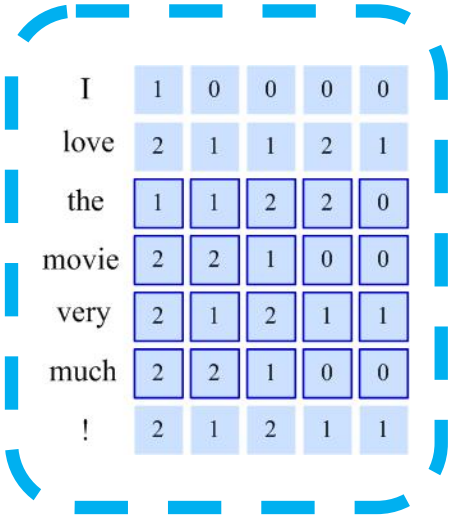
[123, 234, 345, 456, ...]

tf.nn.embedding_lookup

I	1	0	0	0	0
love	2	1	1	2	1
the	1	1	2	2	0
movie	2	2	1	0	0
very	2	1	2	1	1
much	2	2	1	0	0
!	2	1	2	1	1

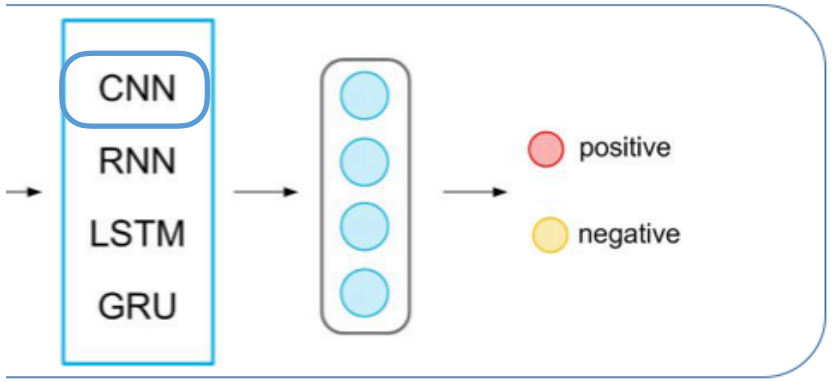
输入向量化

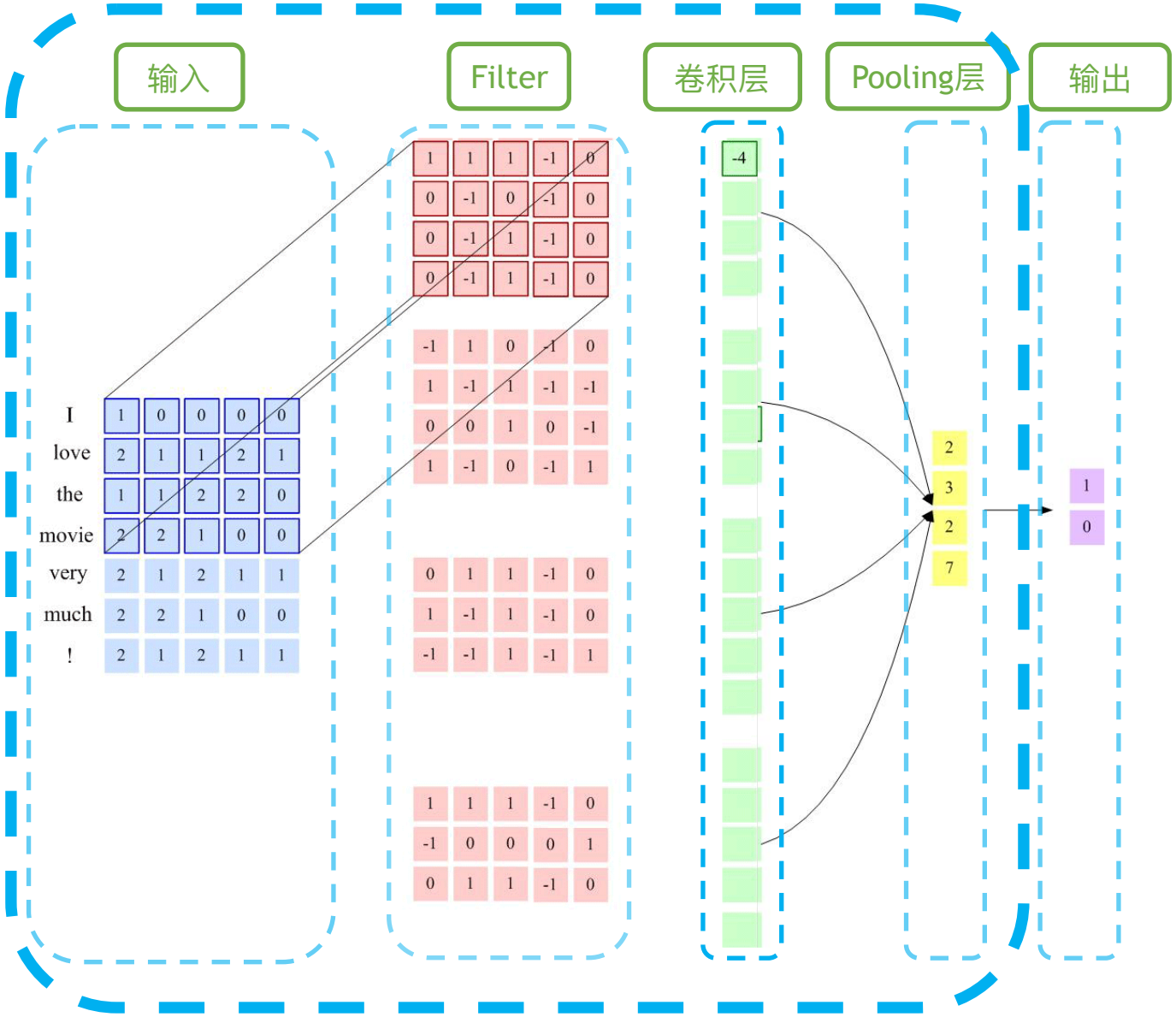




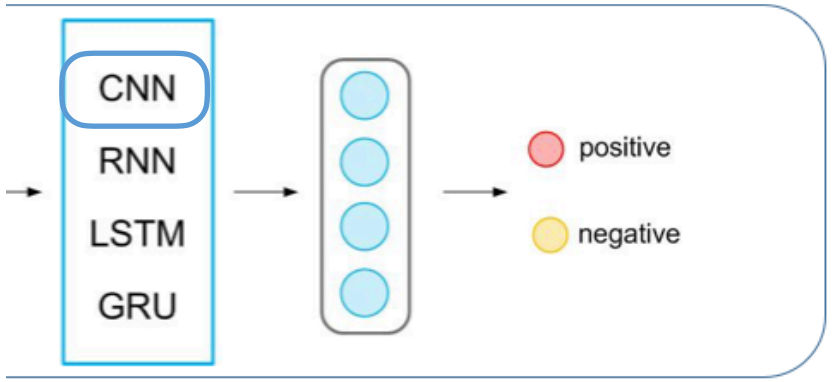

```
ph_input = tf.placeholder(tf.int32, (None, None))
ph_label = tf.placeholder(tf.int32, (None, 1))
ph_seqLen = tf.placeholder(tf.int32, (None,))

embedding = tf.get_variable('Embedding',
            [len(vocab), embed_size],
            trainable=True,
            initializer=tf.random_normal_initializer())
inputs = tf.nn.embedding_lookup(embedding, ph_input)
snt_enc = snt_encoder_cnn(config, inputs, ph_seqLen)
```

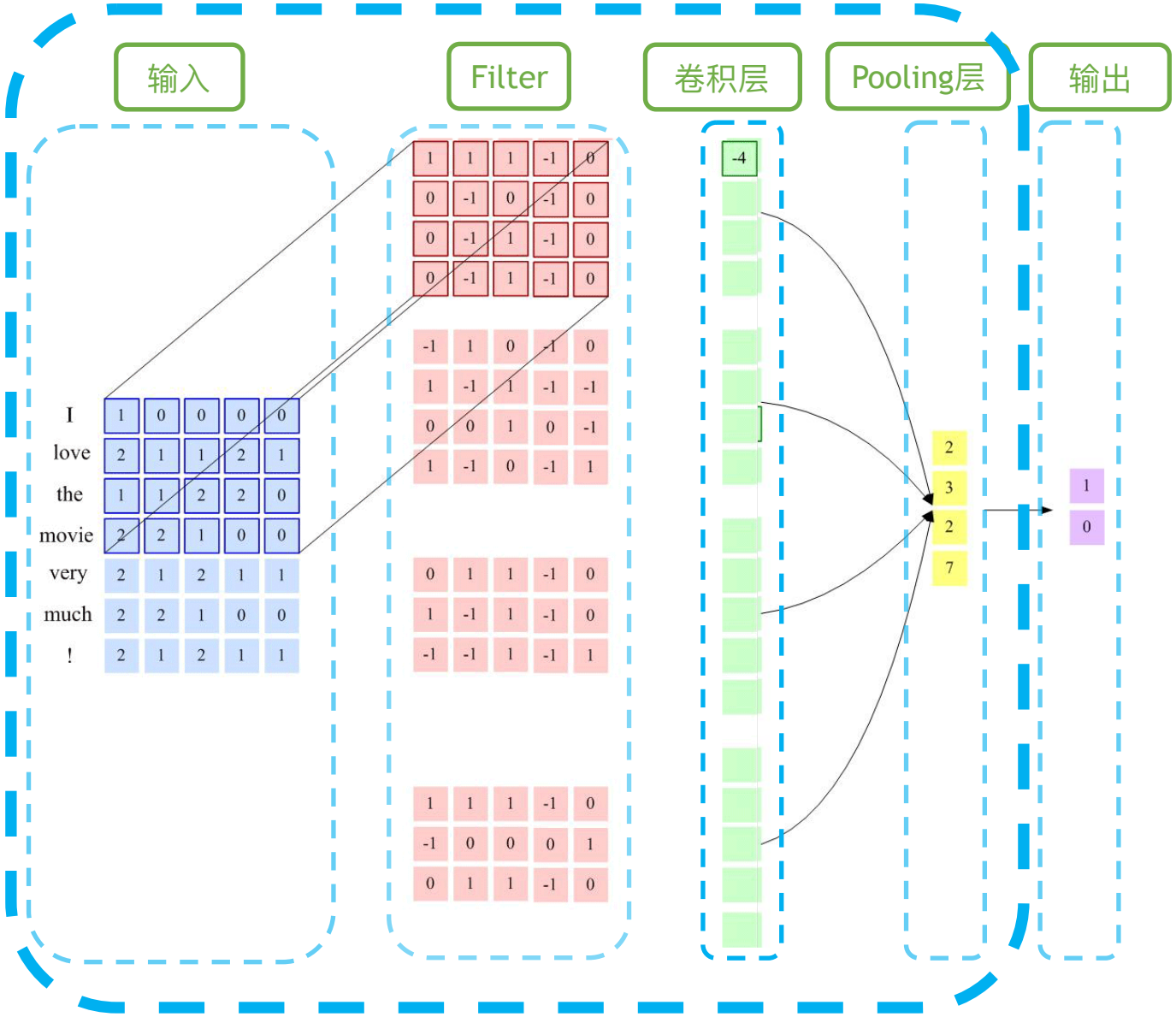
CNN模型框架


CNN
RNN
LSTM
GRU
positive
negative

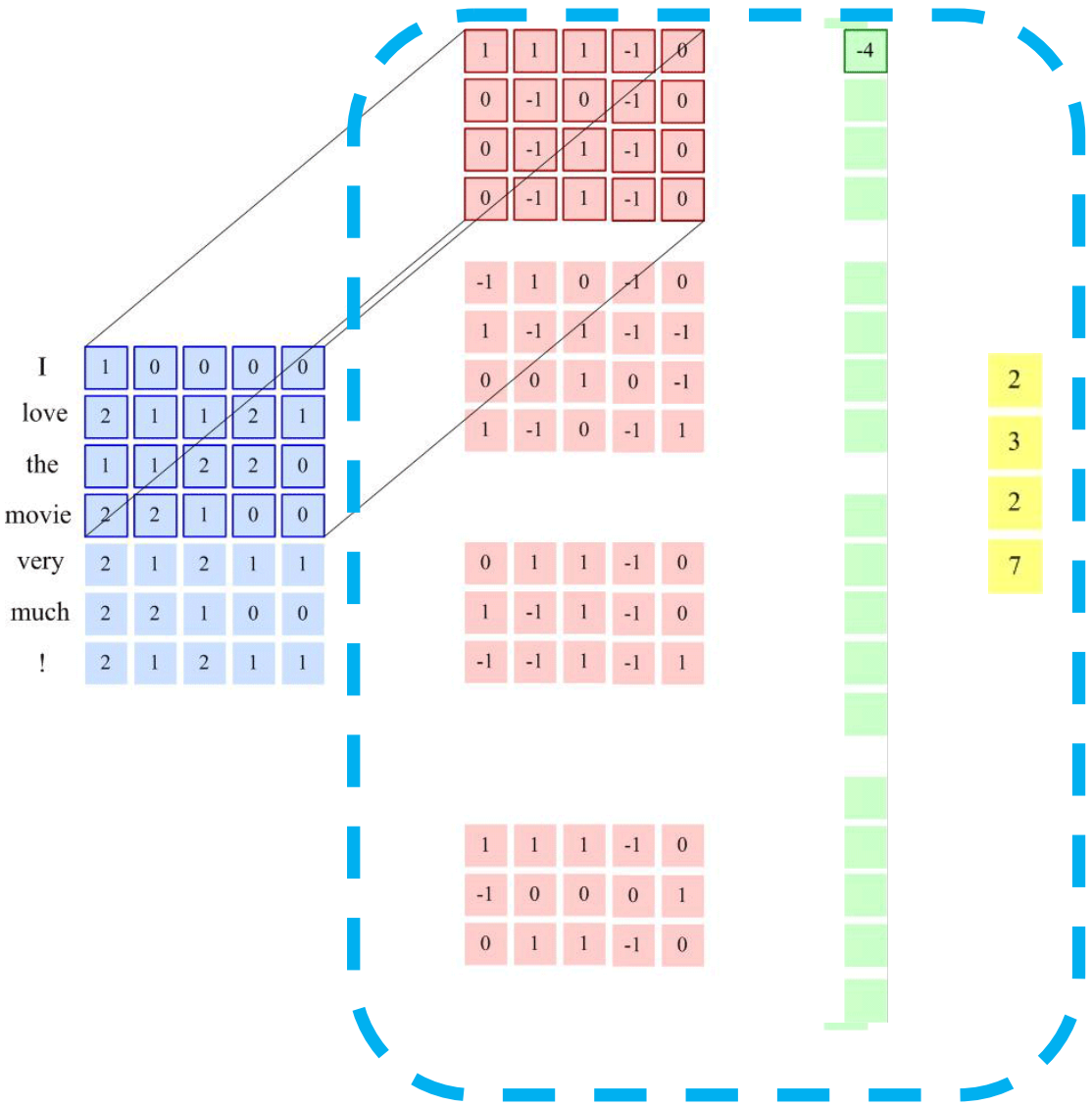

输入
Filter
卷积层
Pooling层
输出
I
love
the
movie
very
much
!

CNN模型框架


CNN
RNN
LSTM
GRU
positive
negative


输入
Filter
卷积层
Pooling层
输出
I
love
the
movie
very
much
!

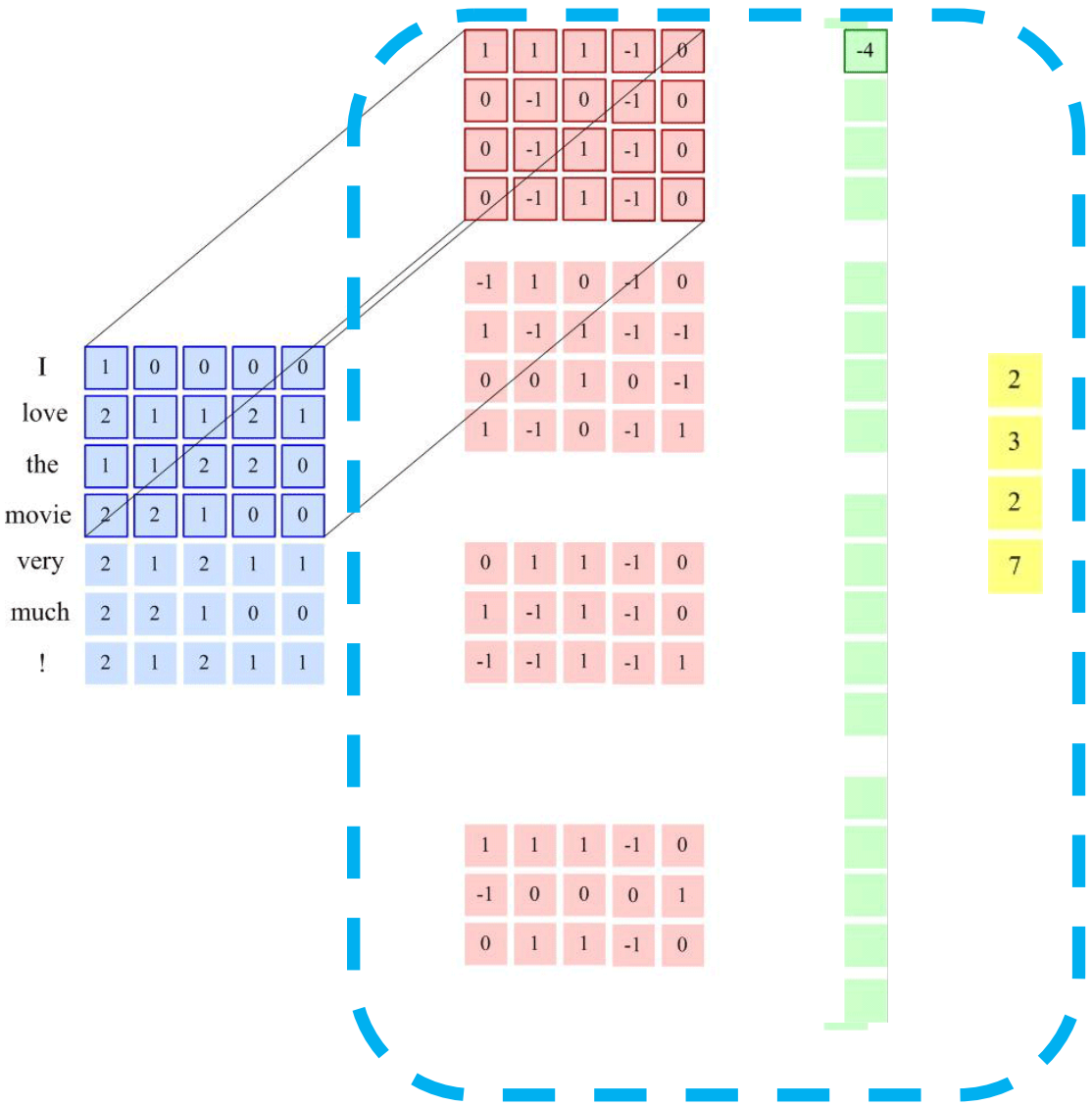
卷积层 + Pooling




```
for i, filter_size in enumerate(filter_sizes):
    with tf.variable_scope("conv-maxpool-%d" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, in_channel, out_channel]
        W = tf.get_variable(name='W', shape=filter_shape)
        b = tf.get_variable(name='b', shape=[out_channel])
        conv = tf.nn.conv1d(  # size (b_sz, tstp, out_channel)
            input,
            W,
            stride=1,
            padding="SAME",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        conv_outputs.append(h)
input = tf.concat(axis=2, values=conv_outputs)  # b_sz, tstp, o
in_channel = out_channel * len(filter_sizes)
```

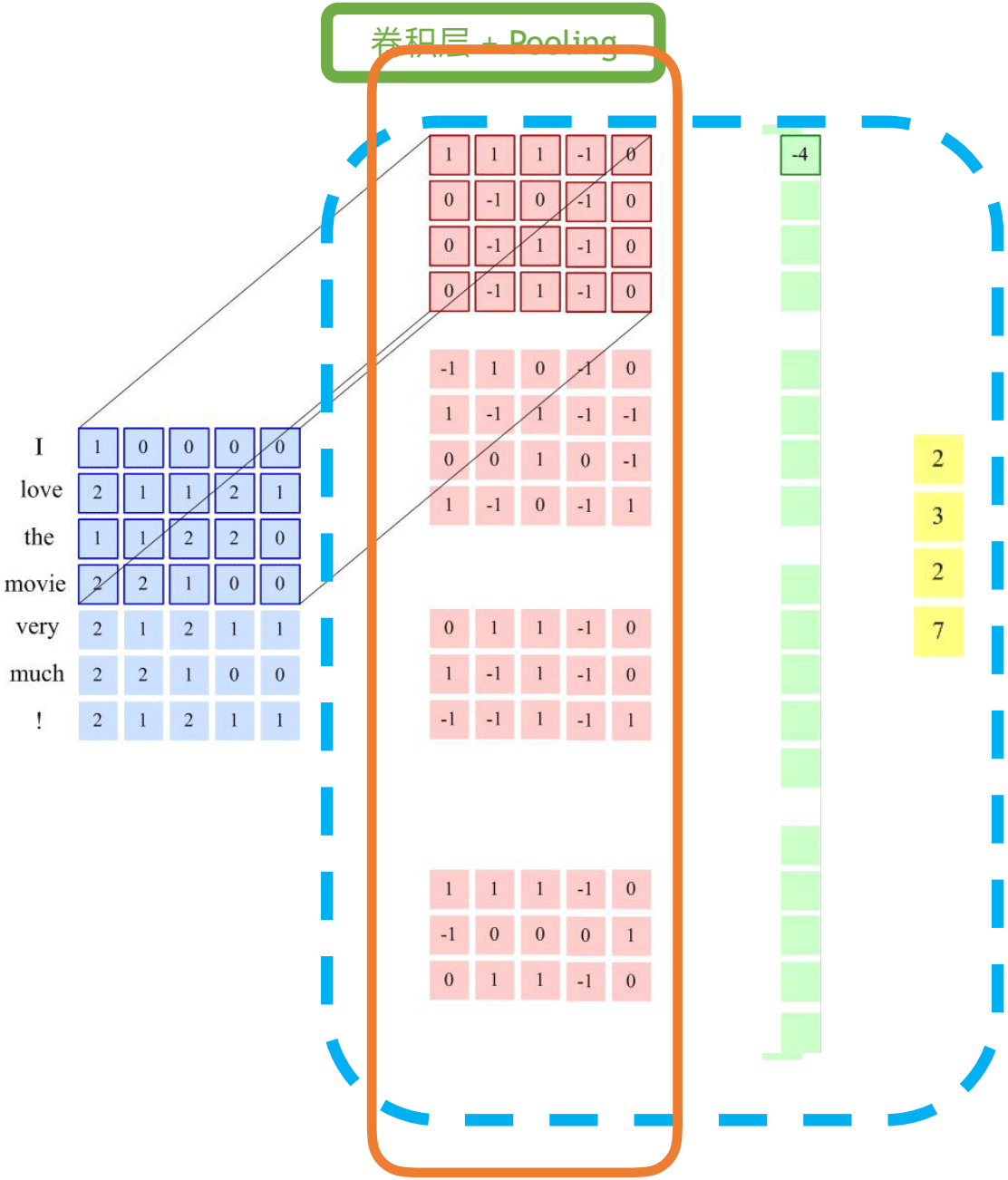
```
pooled = tf.reduce_max(input, axis=1)
# size (b_sz, out_channel*len(filter_sizes))
```

卷积层 + Pooling




```
for i, filter_size in enumerate(filter_sizes):
    with tf.variable_scope("conv-maxpool-%d" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, in_channel, out_channel]
        W = tf.get_variable(name='W', shape=filter_shape)
        b = tf.get_variable(name='b', shape=[out_channel])
        conv = tf.nn.conv1d(  # size (b_sz, tstp, out_channel)
            input,
            W,
            stride=1,
            padding="SAME",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        conv_outputs.append(h)
input = tf.concat(axis=2, values=conv_outputs)  # b_sz, tstp, o
in_channel = out_channel * len(filter_sizes)
```

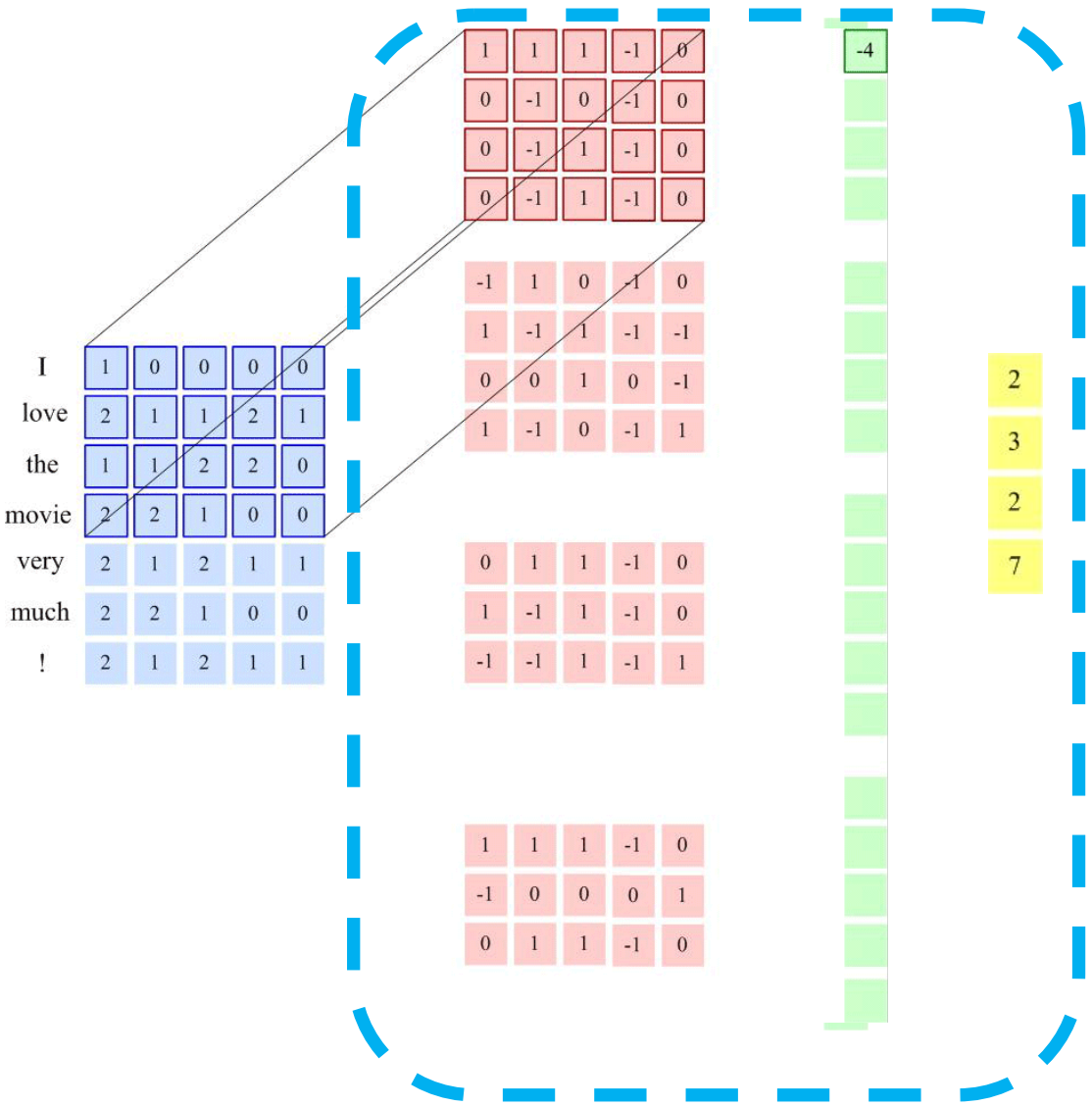
```
pooled = tf.reduce_max(input, axis=1)
# size (b_sz, out_channel*len(filter_sizes))
```




```
for i, filter_size in enumerate(filter_sizes):
    with tf.variable_scope("conv-maxpool-%d" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, in_channel, out_channel]
        W = tf.get_variable(name='W', shape=filter_shape)
        b = tf.get_variable(name='b', shape=[out_channel])
        conv = tf.nn.conv1d(  # size (b_sz, tstp, out_channel)
            input,
            W,
            stride=1,
            padding="SAME",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        conv_outputs.append(h)
input = tf.concat(axis=2, values=conv_outputs)  # b_sz, tstp, o
in_channel = out_channel * len(filter_sizes)
```

```
pooled = tf.reduce_max(input, axis=1)
# size (b_sz, out_channel*len(filter_sizes))
```

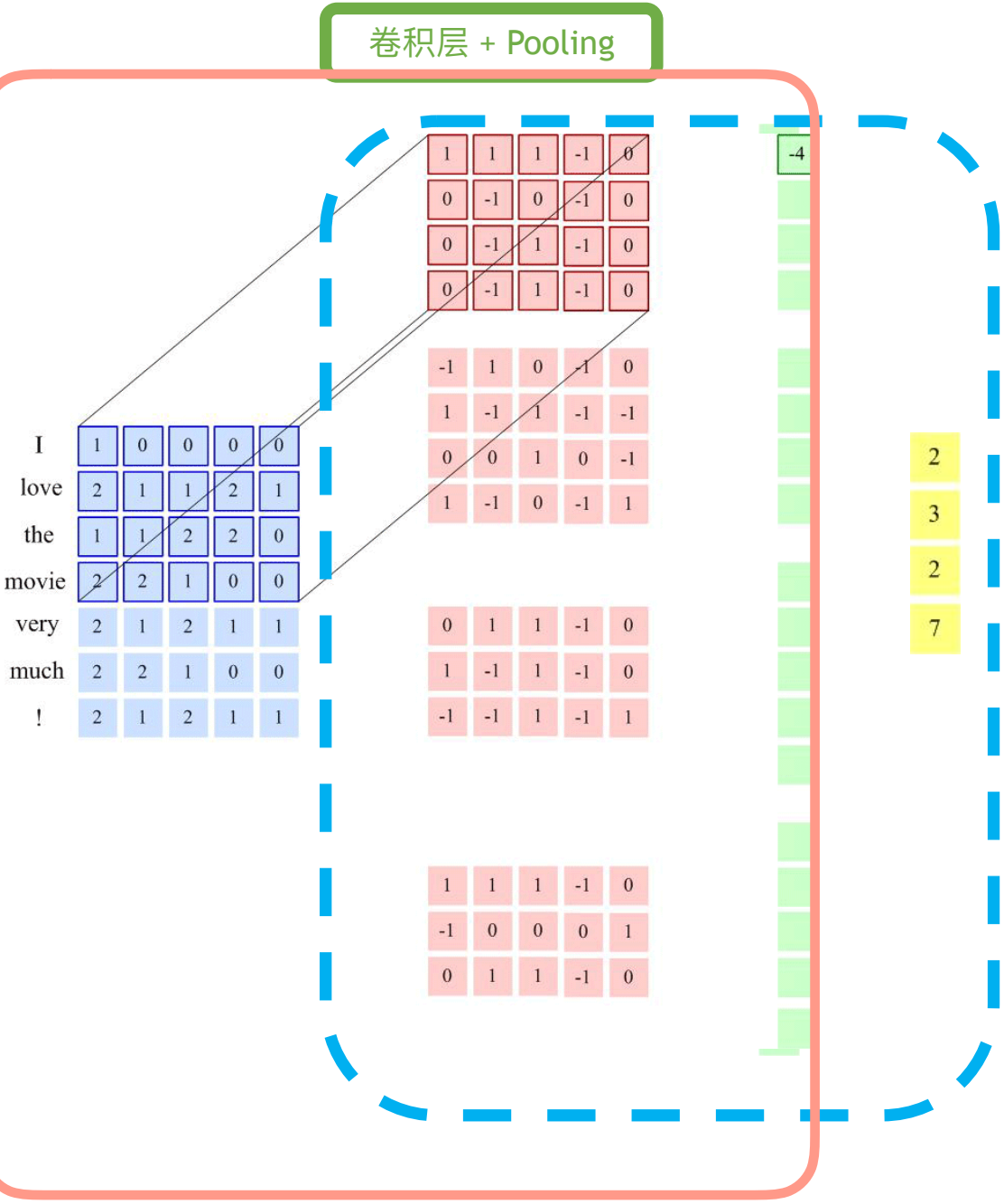
卷积层 + Pooling




```
for i, filter_size in enumerate(filter_sizes):
    with tf.variable_scope("conv-maxpool-%d" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, in_channel, out_channel]
        W = tf.get_variable(name='W', shape=filter_shape)
        b = tf.get_variable(name='b', shape=[out_channel])
        conv = tf.nn.conv1d(  # size (b_sz, tstp, out_channel)
            input,
            W,
            stride=1,
            padding="SAME",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        conv_outputs.append(h)
input = tf.concat(axis=2, values=conv_outputs)  # b_sz, tstp, o
in_channel = out_channel * len(filter_sizes)
```

```
pooled = tf.reduce_max(input, axis=1)
# size (b_sz, out_channel*len(filter_sizes))
```

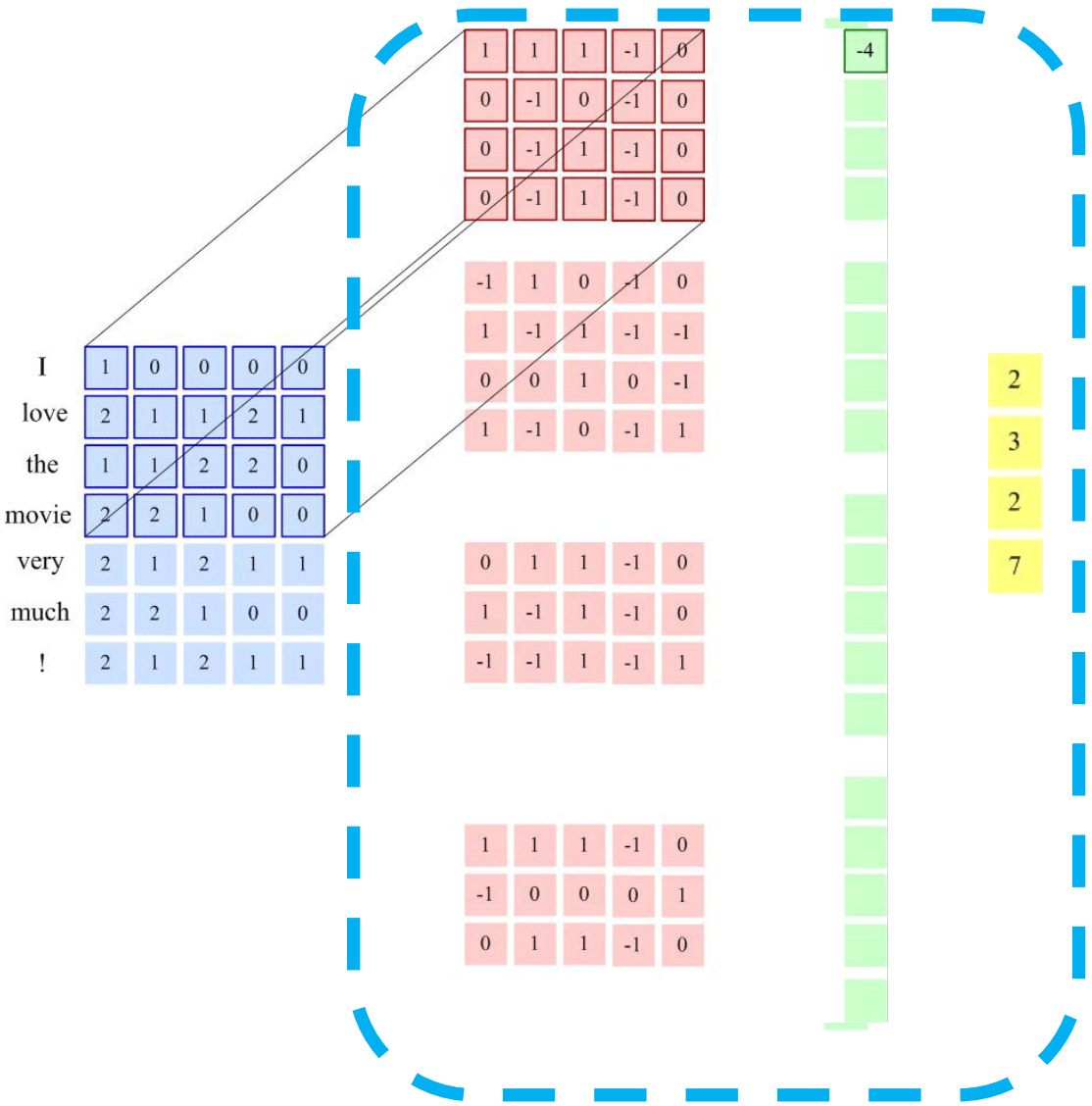
卷积层 + Pooling




```
for i, filter_size in enumerate(filter_sizes):
    with tf.variable_scope("conv-maxpool-%d" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, in_channel, out_channel]
        W = tf.get_variable(name='W', shape=filter_shape)
        b = tf.get_variable(name='b', shape=[out_channel])
        conv = tf.nn.conv1d(  # size (b_sz, tstp, out_channel)
            input,
            W,
            stride=1,
            padding="SAME",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        conv_outputs.append(h)
input = tf.concat(axis=2, values=conv_outputs)  # b_sz, tstp, o
in_channel = out_channel * len(filter_sizes)
```

```
pooled = tf.reduce_max(input, axis=1)
# size (b_sz, out_channel*len(filter_sizes))
```

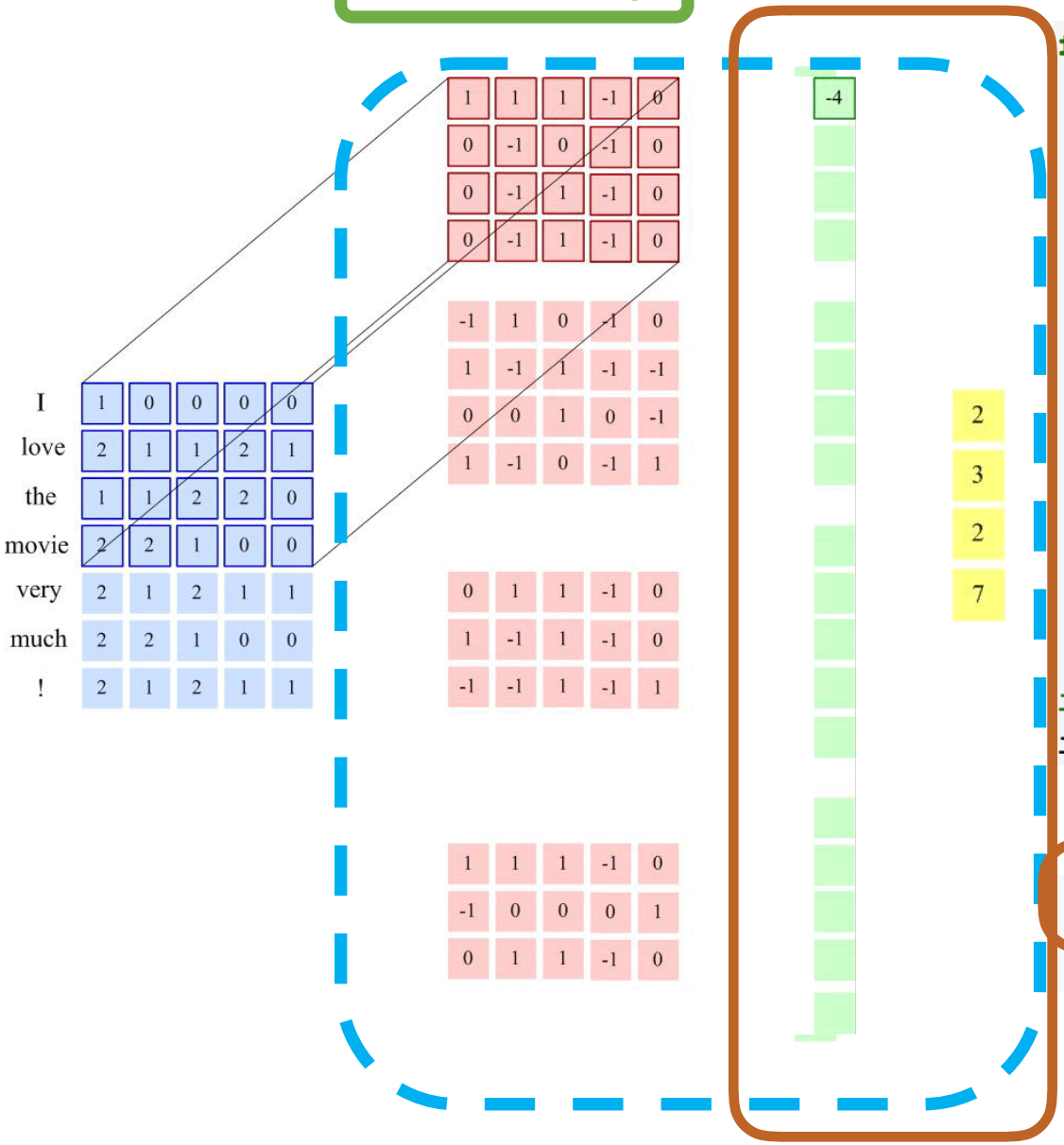
卷积层 + Pooling




```
for i, filter_size in enumerate(filter_sizes):
    with tf.variable_scope("conv-maxpool-%d" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, in_channel, out_channel]
        W = tf.get_variable(name='W', shape=filter_shape)
        b = tf.get_variable(name='b', shape=[out_channel])
        conv = tf.nn.conv1d(  # size (b_sz, tstp, out_channel)
            input,
            W,
            stride=1,
            padding="SAME",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        conv_outputs.append(h)
input = tf.concat(axis=2, values=conv_outputs)  # b_sz, tstp, o
in_channel = out_channel * len(filter_sizes)
```

```
pooled = tf.reduce_max(input, axis=1)
# size (b_sz, out_channel*len(filter_sizes))
```

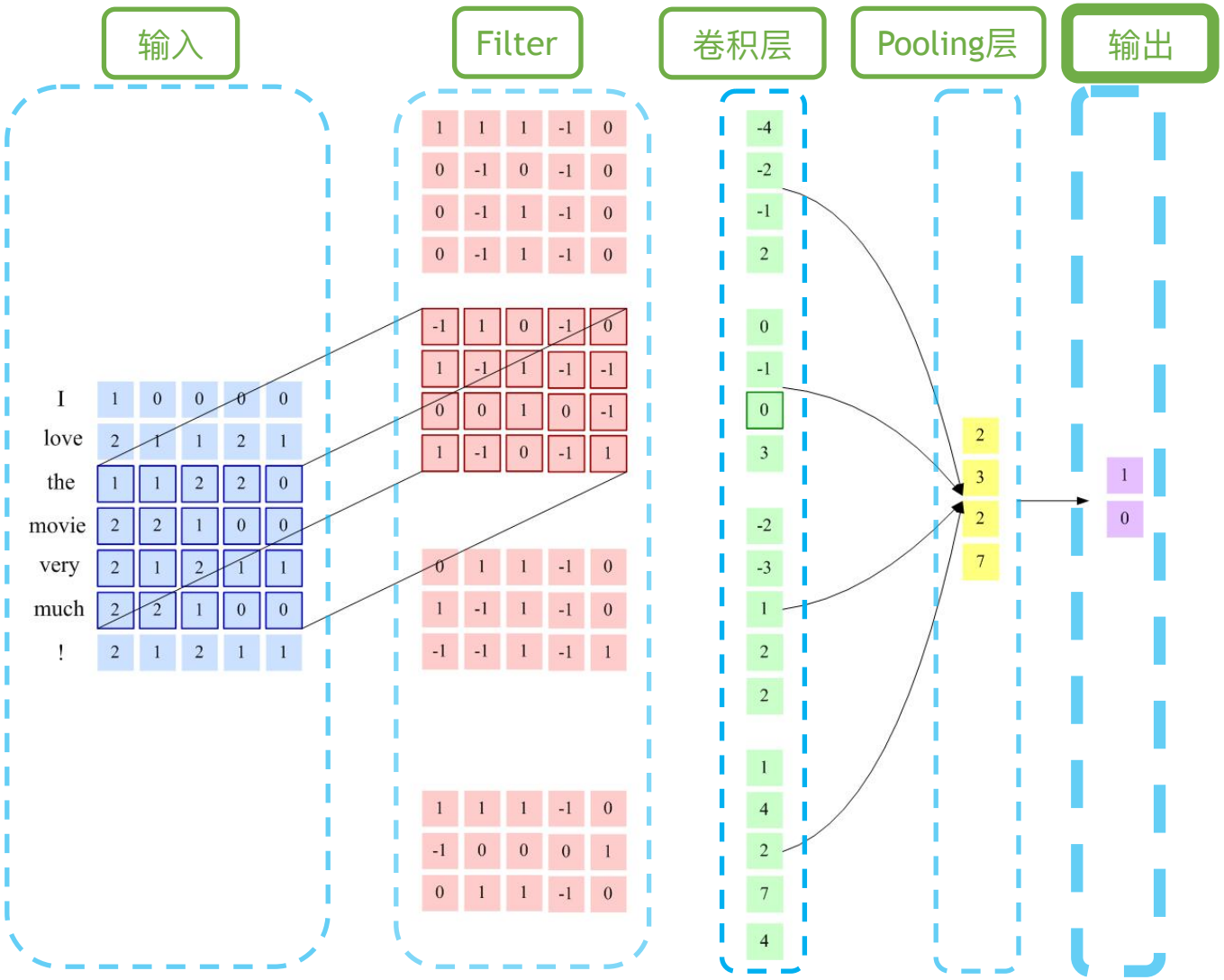
卷积层 + Pooling

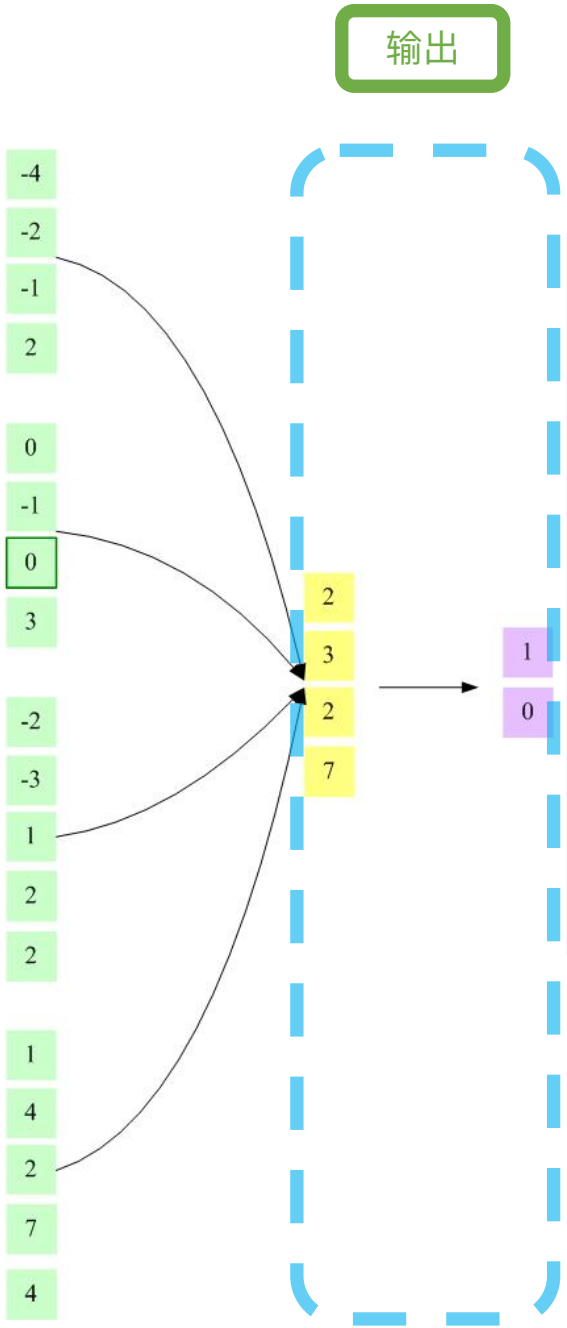



```python
for i, filter_size in enumerate(filter_sizes):
    with tf.variable_scope("conv-maxpool-%d" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, in_channel, out_channel]
        W = tf.get_variable(name='W', shape=filter_shape)
        b = tf.get_variable(name='b', shape=[out_channel])
        conv = tf.nn.conv1d(  # size (b_sz, tstp, out_channel)
            input,
            W,
            stride=1,
            padding="SAME",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        conv_outputs.append(h)
input = tf.concat(axis=2, values=conv_outputs)  # b_sz, tstp, o
in_channel = out_channel * len(filter_sizes)
```

```python
pooled = tf.reduce_max(input, axis=1)
# size (b_sz, out_channel*len(filter_sizes))
```

输出层






```python
def Dense(config, inputs):
    loop_input = inputs
    for i, hid_num in enumerate(config.dense_hidden):
        loop_input = tf.layers.dense(inputs=loop_input,
                                     units=hid_num,
                                     use_bias=True,
                                     scope='dense-layer-%d'%i)
        if i < len(config.dense_hidden)-1:
            loop_input = tf.nn.relu(loop_input)

    logits = loop_input
    return logits
```

```python
prob = tf.nn.softmax(logits)
prediction = tf.arg_max(prob, dimension=1)
```

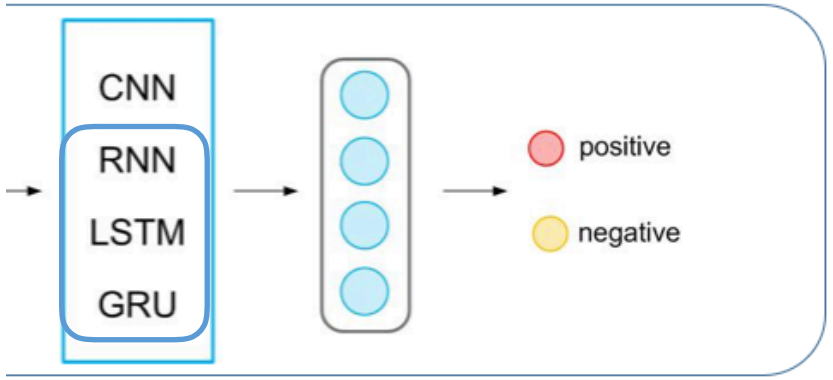
Loss 以及优化

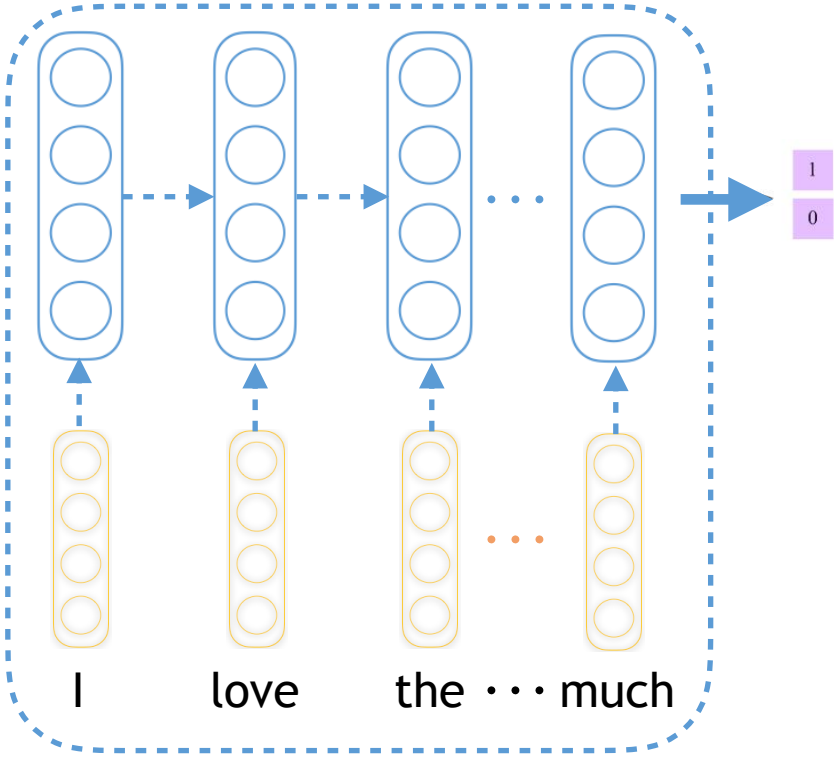
```
def add_loss_op(logits, labels):
    labels_float = tf.to_float(labels)
    loss = tf.nn.softmax_cross_entropy_with_logits(
        logits=logits,
        labels=labels_float)
    loss = tf.reduce_mean(loss)
    return loss

def add_train_op(config, loss):
    global_step = tf.Variable(0, name='global_step',
                              trainable=False)
    learning_rate = tf.train.exponential_decay(
        config.lr, global_step,config.decay_steps,
        config.decay_rate, staircase=True)
    optimizer = tf.train.AdamOptimizer(learning_rate)
    train_op = optimizer.minimize(loss, global_step=global_step)
    return train_op
```

训练

```
with tf.Session() as sess:
    data_len = len(data_x)
    total_steps =data_len // config.batch_size
    for epoch in range(50):
        total_loss = []
        for step, indices in enumerate(
            helper.data_iter_indices(data_len, config.batch_size)):
            feed_dict = create_feed_dict(ph_input, ph_seqLen, ph_label,
                                         data_x[indices],
                                         len_list[indices],
                                         data_y[indices])
            _, loss, lr = sess.run([train_op, loss, learning_rate],
                                   feed_dict=feed_dict)
```

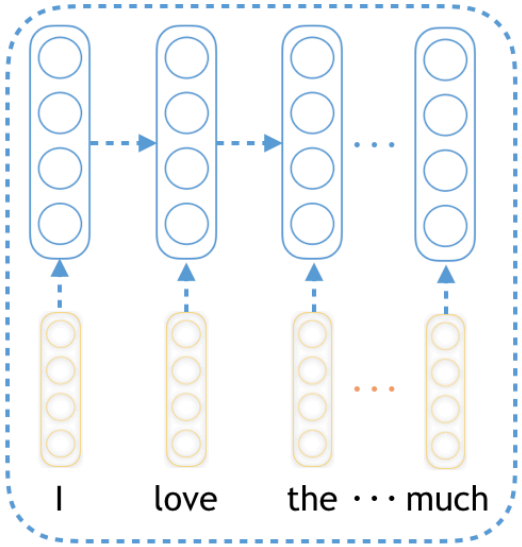
RNN模型框架





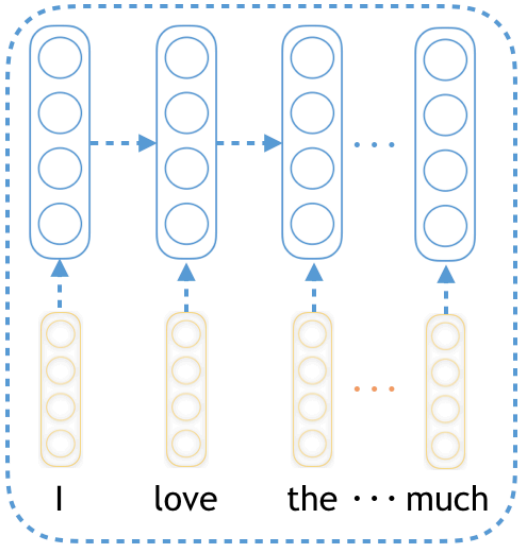
RNN: Basic RNN

```
def snt_encoder_rnn(config, seqInput, seqLen):
    rnn_cell = tf.nn.rnn_cell.BasicRNNCell(config.hidden_size)

    output, states = tf.nn.dynamic_rnn(cell=rnn_cell,inputs=seqInput,
                                       sequence_length=seqLen, dtype=tf.float32,
                                       swap_memory=True, scope='snt_enc')

    snt_enc = states
    return snt_enc
```



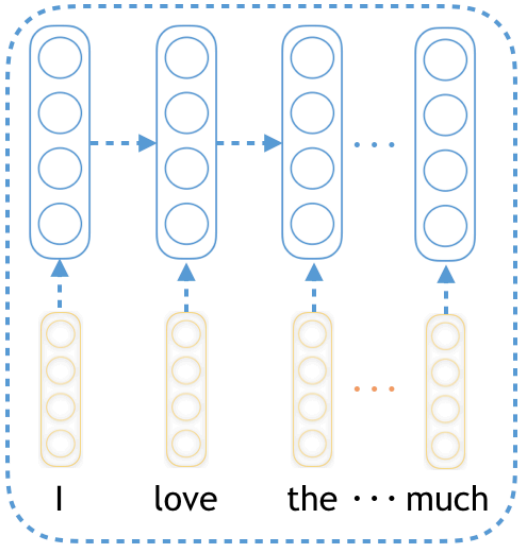
RNN: GRU

```
def snt_encoder_gru(config, seqInput, seqLen):
    gru_cell = tf.nn.rnn_cell.GRUCell(config.hidden_size)

    output, states = tf.nn.dynamic_rnn(cell=gru_cell,inputs=seqInput,
                                       sequence_length=seqLen, dtype=tf.float32,
                                       swap_memory=True, scope='snt_enc')

    snt_enc = states
    return snt_enc
```



RNN: LSTM

```
def snt_encoder_lstm(config, seqInput, seqLen):
    lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(config.hidden_size)

    output, states = tf.nn.dynamic_rnn(cell=lstm_cell,inputs=seqInput,
                                       sequence_length=seqLen, dtype=tf.float32,
                                       swap_memory=True, scope='snt_enc')

    snt_enc = states[1]
    return snt_enc
```



LSTM 简单实现

```
def lstm_step(in_x, hidden_state, cell_state,
              hidden_size, emb_size):

    in_x_h = tf.concat([in_x, hidden_state])
    W = tf.get_variable(name='W', shape=[emb_size+hidden_size,
                                         4*hidden_size])
    b = tf.get_variable(name='b', shape=[4*hidden_size])
    concat = tf.nn.bias_add(tf.matmul(in_x_h, W), b)
    i, j, f, o = tf.split(concat, num_or_size_splits=4, axis=1)
    next_cell_state = (cell_state * sigmoid(f)
                       + sigmoid(i) * tf.tanh(j))
    next_h = tf.tanh(next_cell_state) * sigmoid(o)

    return next_h, next_cell_state
```

LSTM 简单实现

```
def rnn(inputs, step, hidden_size, embed_size,
        init_cell_state, init_hidden):

    hidden = init_hidden
    state = init_cell_state
    output = []

    with tf.variable_scope('rnn') as varscope:
        for i in range(step):
            if i==1:
                varscope.reuse_variables()
            hidden, state = lstm_step(inputs[i], hidden, state,
                                      hidden_size, embed_size)
            output.append(hidden)
    return output
```

Examples

- Logistic regression
- Linear regression
- Example on Github: https://github.com/JerrikEph/cips2017_tfTutorial

谢谢各位各位老师同学☺️