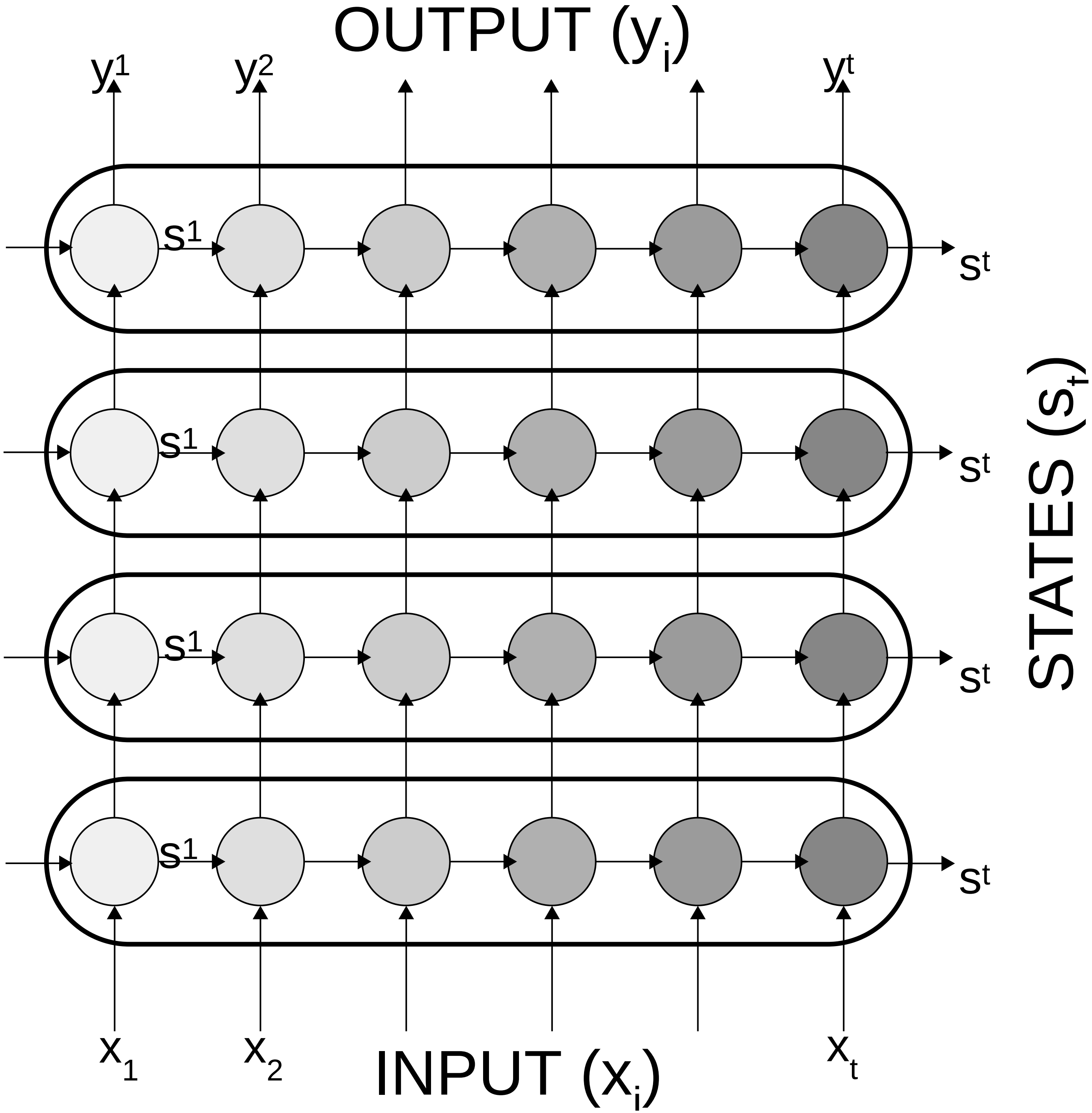
Multi RNN

Practical explanation

An input sequence into a single RNN, that's contructed from many cells (e.g. LSTM), that has tensor shape `[batch_size, max_time, number_of_features]` will result to an output of `[batch_size, max_time, hidden_dim]` . See notebook from `Chapter 10 RNN` . When we stack multiple RNNs (see image below) our output/input relationship is the same as a single RNN but our model is more complex and it can learn more complex tasks. We need this complexity for seq2seq models.



OUTPUT (y_i)
y^1
y^2
y^t
s^1
s^1
s^1
s^1
s^t
s^t
s^t
s^t
STATES (s_t)
x_1
x_2
x_t
INPUT (x_i)

Code

All we need is TensorFlow:

In [1]:
```
import tensorflow as tf
import numpy as np
```

First, define the constants.

Let's say we're dealing with 1-dimensional vectors, and a maximum sequence size of 3.

In [2]:
```
input_seq = [[1,2], [2,3], [3,4]]
print('Input shape = {}'.format(np.array(input_seq).shape))
```

```
Input shape = (3, 2)
```

In [3]:
```
input_dim = np.array(input_seq).shape[1]#2
seq_size = np.array(input_seq).shape[0]
```

Next up, define the placeholder(s).

We only need one for this simple example: the input placeholder.

The placeholder has `None` as the first dimension since we can have variable batch sizes

In [4]:
```
input_placeholder = tf.placeholder(dtype=tf.float32, shape=[None, seq_size, input_dim])
```

Function to create one LSTM cell

In [5]:
```
def make_cell(state_dim):
    return tf.contrib.rnn.LSTMCell(state_dim)
```

Call the function and extract the cell outputs.

In [6]:
```
with tf.variable_scope("first_cell") as scope:
    cell = make_cell(state_dim=10)
    outputs, states = tf.nn.dynamic_rnn(cell, input_placeholder, dtype=tf.float32)
```

Create a new RNN that takes the output of the first-cell RNN

In essense stacking one on top of the other

In [7]:
```
with tf.variable_scope("second_cell") as scope:
    cell2 = make_cell(state_dim=10)
    outputs2, states2 = tf.nn.dynamic_rnn(cell2, outputs, dtype=tf.float32)
```

We can create a list of cells and stack them ontop of each other and then the `MultiRNNCell` to create a layered RNN for multiple cells

In [8]:
```
def make_multi_cell(state_dim, num_layers):
    cells = [make_cell(state_dim) for _ in range(num_layers)]
    return tf.contrib.rnn.MultiRNNCell(cells)
multi_cell = make_multi_cell(state_dim=10, num_layers=5)
outputs5, states5 = tf.nn.dynamic_rnn(multi_cell, input_placeholder, dtype=tf.float32)
```

Before starting a session, let's prepare some simple input to the network.

Run session

In [9]:

```python
with tf.Session() as sess:
    # initialise
    sess.run(tf.global_variables_initializer())
    sess.run(tf.local_variables_initializer())
    outputs_val, outputs2_val, outputs5_val = sess.run([outputs, outputs2, outputs5],
                                                        feed_dict={input_placeholder: [input_seq]})
    states_val, states2_val, states5_val = sess.run([states, states2, states5],
                                                        feed_dict={input_placeholder: [input_seq]})
print('*** Shape of single cell output ****')
print('Dimensions of output [batch_size, time_step, hidden_units]-> {}'.format(outputs_val.shape))
print('*** Shape of multiple state output ****')
print('Dimensions of output [batch_size, time_step, hidden_units]-> {}'.format(outputs5_val.shape))
print('***Single Cell States Shape****')
print('********************************************')
print('Hidden state shape = {0}'.format(states_val[0].shape))
print('Activation state shape = {0}'.format(states_val[1].shape))
print('***Multiple Cell States Shape****')
print('********************************************')
for i, state in enumerate(states5_val):
    print('Layer {0} of hidden state shape = {1}'.format(i+1, state[0].shape))
    print('Layer {0} of activation state shape = {1}'.format(i+1, state[1].shape))
    print('********************************************')
```

```
*** Shape of single cell output ****
Dimensions of output [batch_size, time_step, hidden_units]-> (1, 3, 10)
*** Shape of multiple state output ****
Dimensions of output [batch_size, time_step, hidden_units]-> (1, 3, 10)
***Single Cell States Shape****
********************************************
Hidden state shape = (1, 10)
Activation state shape = (1, 10)
***Multiple Cell States Shape****
********************************************
Layer 1 of hidden state shape = (1, 10)
Layer 1 of activation state shape = (1, 10)
********************************************
Layer 2 of hidden state shape = (1, 10)
Layer 2 of activation state shape = (1, 10)
********************************************
Layer 3 of hidden state shape = (1, 10)
Layer 3 of activation state shape = (1, 10)
```

```
*******************************************
Layer 4 of hidden state shape = (1, 10)
Layer 4 of activation state shape = (1, 10)
*******************************************
Layer 5 of hidden state shape = (1, 10)
Layer 5 of activation state shape = (1, 10)
*******************************************
```