Linear Regression with Multiple Variables in TensorFlow

In Andrew Ng shows how to generalize linear regression with a single variable to the case of multiple variables. Andrew Ng introduces a bit of notation to derive a more succinct formulation of the problem. Namely, n features $x_1 \ldots . x_n$ are extended by adding feature $x_0$which is always set to 1. This way the hypothesis can be expressed as:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \theta^T x$$

For m examples, the task of linear regression can be expressed as a task of finding vector θ such that

$$\begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix} \times \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ & & \vdots & \\ x_m^{(n)} & x_m^{(n)} & \cdots & x_m^{(n)} \end{bmatrix}$$

is as close as possible to some observed values $y_1, y_1, \ldots\ldots, y_m$. The “as close as possible” typically means that the mean sum of square errors between $h_\theta(x^{(i)})$ and y_i for $i \in [1, m]$ is minimized. This quantity is often referred to as cost or loss function:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y_i\right)^2$$

To express the above concepts in TensorFlow, and more importantly, have TensorFlow find θ that minimizes the cost function, we need to make a few adjustments. We rename vector θ, as w. We are not using $x_0 = 1$. Instead, we use a tensor of size 0 (also known as scalar), called b to represent x_0 . As

it is easier to stack rows than columns, we form matrix X, in such a way that the i-th row is the i-th sample. Our formulation thus has the form

$$h_{w,b}(X) = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix} \times \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} + b$$

This leads to the following Python code:

```
n=321
# Placeholder that is fed input data.
x = tf.placeholder(tf.float32, [None, n], "X_in")

# The model: we assume y = X_in * w + b
w = tf.Variable(tf.random_normal([n, 1]), name="w")
b = tf.Variable(tf.constant(1.0,shape=[]), name="b")
h = tf.add(tf.matmul(x, w), b, name="h")
```

We first introduce a tf.placeholder named X_in. This is how we supply data into our model. Line 2 creates a vector w corresponding to θ.Line 3 creates a variable b corresponding to θ . Finally, line 4 expresses function h as a matrix multiplication of X_in and w plus scalar b

```
Y = tf.placeholder(tf.float32, [None, 1], "y_in")

# The loss function: we are minimizing square root of mean
loss_op = tf.reduce_mean(tf.square(tf.subtract(Y, h)), name="loss")
train_op = tf.train.AdamOptimizer(learning_rate).minimize(loss_op)
```

To define the loss function, we introduce another placeholder y_in. It holds the ideal (or target) values for the function h. Next, we create a loss_op. This corresponds to the loss function. The difference is that, rather than being a

function directly, it defines for TensorFlow operations that need to be run to compute a loss function. Finally, the training operation uses a gradient descent optimizer, that uses learning rate of 0.3, and tries to minimize the loss.
Now we have all pieces in place to create a loop that finds w and b that minimize the loss function.

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for step in range(2000):
        sess.run(train_op, feed_dict={
          x: np.asarray(X_train),
          Y: np.asarray(y_train)
        })
    w_computed = sess.run(w)
    b_computed = sess.run(b)
```