Linear Regression

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

Let's generate some synthetic data. Our target function is $1x_1 + 2x_2 + 3x_3 + 4x_4$

In [2]:
```
a = np.array([[1, 2, 3, 4]], dtype=np.float32)
XX = np.random.rand(10000, 4)
noise = (np.random.rand(10000, 4) - 0.5) / 1000
YY = np.dot(XX+noise, a.transpose())
```

Let's assume a very simple linear model. We have four features and we are hoping to predict an outcome with that.

Let's capture this model and predictor

In [3]:
```
# Our data placeholders
X = tf.placeholder(tf.float32, [None, 4])
y = tf.placeholder(tf.float32, [None, 1])

# Our model
w = tf.Variable(tf.random_uniform([4, 1]))

# Predictor
def f(X):
    return tf.matmul(X, w)
```

Now let's capture our objective function

In [6]:
```
def objective(X, y):
    return tf.reduce_sum(tf.square(tf.subtract(y, f(X))))
```

Why don't we let tensorflow do the magic of automatic differentiation

In [7]:
```
gradients = tf.gradients(objective(X,y), [w])
```

Let's build our gradient descent algorithm and see how our algorithm performs

In [11]:
```
# Learning step for our descent
step = tf.constant(1e-5)


with tf.Session() as session:
    session.run(tf.global_variables_initializer())

    print("Starting with a random model:")
    print(session.run(w))

    # Now learn
    for i in range(500):
        session.run(tf.assign_add(w, tf.multiply(-step, gradients[0]))
, feed_dict={X:XX, y:YY})

    # Here's our learned model
    print("Trained model")
    print(session.run(w))
```

```
Starting with a random model:
[[0.5338298 ]
 [0.51265657]
 [0.20189261]
 [0.19267201]]
Trained model
[[1.0004953]
 [2.0001202]
 [2.9997985]
 [3.999631 ]]
```

That's slow!! But we can do better

Let's use a faster optimizer from TF

In [12]:
```
optimizer = tf.train.GradientDescentOptimizer(step)
train = optimizer.minimize(objective(X, y), var_list=[w])

with tf.Session() as session:
    session.run(tf.global_variables_initializer())

    print("Starting with a random model:")
    print(session.run(w))

    # Learning
    for i in range(1000):
        session.run(train, feed_dict={X:XX, y:YY})


    #Our model
    print("Trained model")
    print(session.run(w))
```

```
Starting with a random model:
[[0.54813695]
 [0.8609539 ]
 [0.7238796 ]
 [0.82576776]]
Trained model
[[1.0000949]
 [1.9999945]
 [2.9999297]
 [4.0000253]]
```

Let's look at this on tensorboard

In [14]:
```
writer = tf.summary.FileWriter("./tmp/lin_reg", session.graph)
```

In [15]:
```
accuracy = tf.reduce_mean(100 - tf.abs(y - f(X)))

with tf.Session() as session:
    session.run(tf.global_variables_initializer())

    print("Accuracy is ")
    print(session.run(accuracy, feed_dict={X:XX, y:YY}))
```

```
Accuracy is
96.13379
```

Lets do this as we train

In [16]:
```
tf.summary.scalar('accuracy', accuracy)
merged = tf.summary.merge_all()

with tf.Session() as session:
    session.run(tf.global_variables_initializer())

    # Learning
    for i in range(1000):

        summary, _ = session.run([merged, train], feed_dict={X:XX, y:Y
Y})
        writer.add_summary(summary, i)
```

In [ ]:
```
```