Chapter_3_Section_2_Polynomial_Regression

February 2, 2019

1 Ch 03: Concept 02

1.1 Polynomial regression

Import the relevant libraries and initialize the hyper-parameters

```
In [1]: %matplotlib inline
        import tensorflow as tf
        import numpy as np
        import matplotlib.pyplot as plt
        np.random.seed(1234)
        learning_rate = 0.01
        training_epochs = 40
```

Set up some fake raw input data

```
In [2]: trX = np.linspace(-1, 1, 101)
```

Set up raw output data based on a degree 6 polynomial

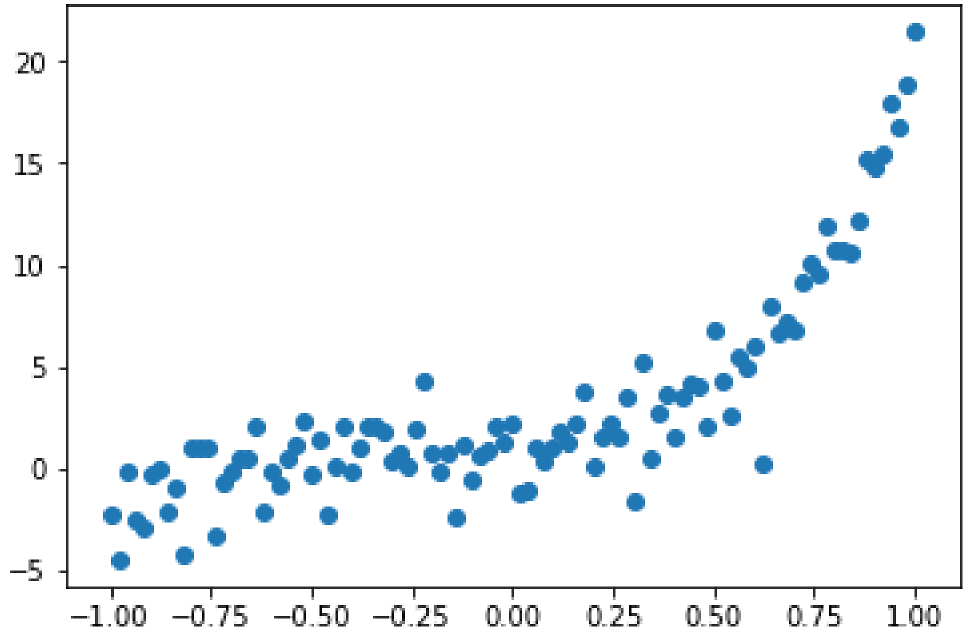
```
In [3]: num_coeffs = 6
        trY_coeffs = [1, 2, 3, 4, 5, 6]
        trY = 0
        for i in range(num_coeffs):
            trY += trY_coeffs[i] * np.power(trX, i)
```

Add some noise

```
In [4]: trY += np.random.randn(*trX.shape) * 1.5
```

Plot the raw data

```
In [5]: plt.scatter(trX, trY)
        plt.show()
```



Define the nodes to hold values for input/output pairs

```
In [6]: X = tf.placeholder("float")
        Y = tf.placeholder("float")
```

Define our polynomial model

```
In [7]: def model(X, w):
            terms = []
            for i in range(num_coeffs):
                term = tf.multiply(w[i], tf.pow(X, i))
                terms.append(term)
            return tf.add_n(terms)
```

Set up the parameter vector to all zero

```
In [8]: w = tf.Variable([0.] * num_coeffs, name="parameters")
        y_model = model(X, w)
```

Define the cost function just as before

```
In [9]: cost = tf.reduce_sum(tf.square(Y-y_model))
        train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Set up the session and run the learning algorithm just as before

```
In [10]: sess = tf.Session()
         init = tf.global_variables_initializer()
         sess.run(init)

         for epoch in range(training_epochs):
             for (x, y) in zip(trX, trY):
                 sess.run(train_op, feed_dict={X: x, Y: y})

         w_val = sess.run(w)
         print(w_val)
```

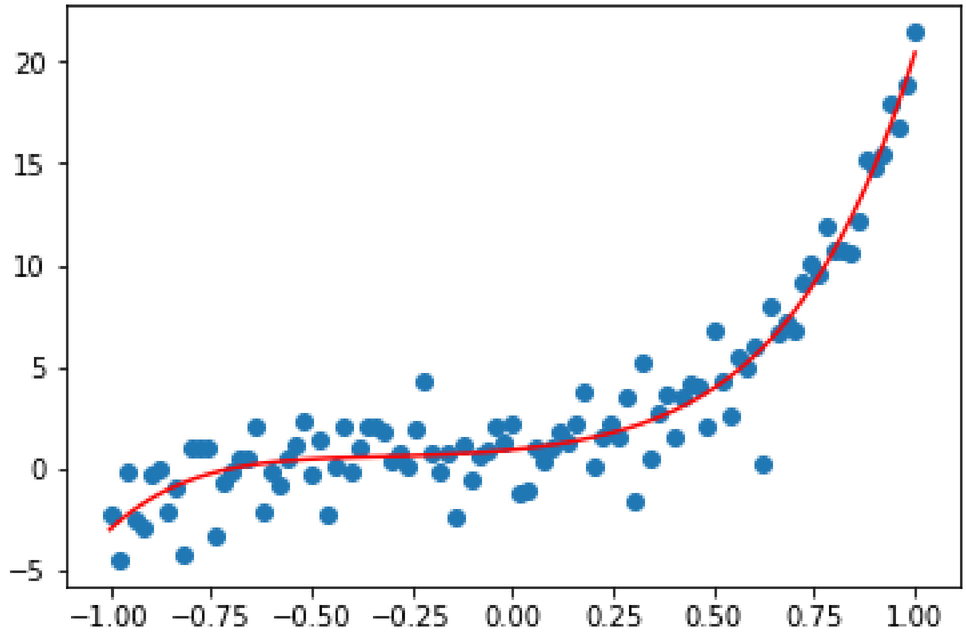
```
[0.93859625 1.962799   4.2336397  4.849198   3.5570402  4.893544  ]
```

Close the session when done

```
In [11]: sess.close()
```

Plot the result

```
In [12]: plt.scatter(trX, trY)
         trY2 = 0
         for i in range(num_coeffs):
             trY2 += w_val[i] * np.power(trX, i)
         plt.plot(trX, trY2, 'r')
         plt.show()
```


20
15
10
5
0
−5
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00