Tarea8_Learning_Slowdown

December 14, 2017

A continuación se presenta un ejemplo del tutorial de TensorFlow para ilustrar el fenómeno de "Learning Slowdown". Este fenómeno consiste en una desaceleración de la tasa de aprendizaje de una red neuronal a medida que el error en la predicción se va disminuyendo durante el proceso de aprendizaje.

El ejemplo del tutorial utiliza una red neuronal con 256 neuronas y 2 capas internas. Se itera 1000 veces y se miden los parámetros de la función de costo y los parámetros de precisión del modelo. Después, se comenta el fenómeno.

A continuación, el código.

```
In [2]: from __future__ import print_function
        # Import MNIST data
        from tensorflow.examples.tutorials.mnist import input_data
        mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
        import tensorflow as tf
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
```

```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

```
In [9]: # Parameters
        learning_rate = 0.1
        num_steps = 1000
        batch_size = 128
        display_step = 100

        # Network Parameters
        n_hidden_1 = 256 # 1st layer number of neurons
        n_hidden_2 = 256 # 2nd layer number of neurons
        num_input = 784 # MNIST data input (img shape: 28*28)
        num_classes = 10 # MNIST total classes (0-9 digits)

        # tf Graph input
```

```
X = tf.placeholder("float", [None, num_input])
Y = tf.placeholder("float", [None, num_classes])
```

In [4]:
```
# Store layers weight & bias
weights = {
    'h1': tf.Variable(tf.random_normal([num_input, n_hidden_1])),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_hidden_2, num_classes]))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([num_classes]))
}
```

In [10]:
```
# Create model
def neural_net(x):
    # Hidden fully connected layer with 256 neurons
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    # Hidden fully connected layer with 256 neurons
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    # Output fully connected layer with a neuron for each class
    out_layer = tf.matmul(layer_2, weights['out']) + biases['out']
    return out_layer
```

In [11]:
```
# Construct model
logits = neural_net(X)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)

# Evaluate model (with test logits, for dropout to be disabled)
correct_pred = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
```

In [12]:
```
# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)
    step_m1=[]
    loss_m1=[]
    accuracy_m1=[]
```

```
    for step in range(1, num_steps+1):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop)
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y})
        if step % display_step == 0 or step == 1:
            # Calculate batch loss and accuracy
            loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x,
                                                                 Y: batch_y})
            print("Step " + str(step) + ", Minibatch Loss= " + \
                  "{:.4f}".format(loss) + ", Training Accuracy= " + \
                  "{:.3f}".format(acc))

            step_m1.append(step)
            loss_m1.append(loss)
            accuracy_m1.append(acc)

    print("Optimization Finished!")

    # Calculate accuracy for MNIST test images
    print("Testing Accuracy:", \
        sess.run(accuracy, feed_dict={X: mnist.test.images,
                                      Y: mnist.test.labels}))
```

```
Step 1, Minibatch Loss= 11005.1758, Training Accuracy= 0.430
Step 100, Minibatch Loss= 326.4753, Training Accuracy= 0.797
Step 200, Minibatch Loss= 152.0086, Training Accuracy= 0.867
Step 300, Minibatch Loss= 71.5822, Training Accuracy= 0.898
Step 400, Minibatch Loss= 76.9592, Training Accuracy= 0.859
Step 500, Minibatch Loss= 64.7189, Training Accuracy= 0.789
Step 600, Minibatch Loss= 51.2830, Training Accuracy= 0.852
Step 700, Minibatch Loss= 60.2751, Training Accuracy= 0.867
Step 800, Minibatch Loss= 55.3867, Training Accuracy= 0.883
Step 900, Minibatch Loss= 67.1177, Training Accuracy= 0.852
Step 1000, Minibatch Loss= 52.5904, Training Accuracy= 0.820
Optimization Finished!
Testing Accuracy: 0.8363
```

Notar que en ambos casos desde las iteraciones iniciales disminuye la pérdida del modelo e incrementa la precisión. La precisión en algunas instancias disminuye, pero la mayoria de las veces se mantiene constante, mientras que el costo se mantiente constante desde la iteración 400.

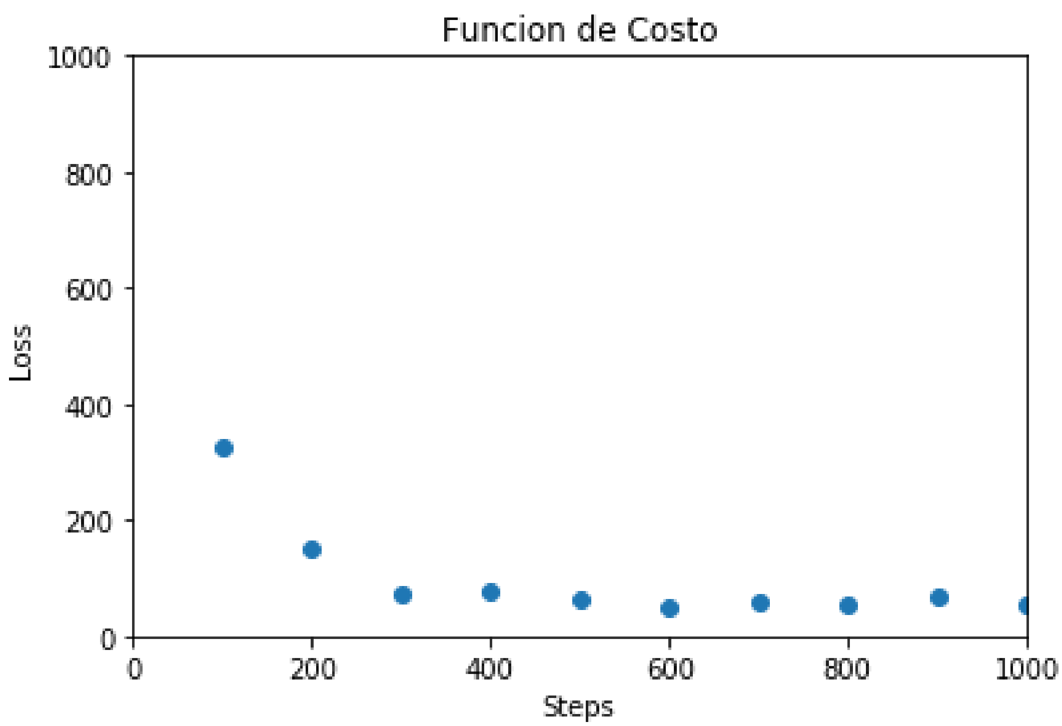
```
In [14]: res_m1=pd.DataFrame(step_m1, columns=['step'])
         res_m1['loss']=loss_m1
         res_m1['acc']=accuracy_m1
         res_m1
```

Out[14]:

	step	loss	acc
0	1	11005.175781	0.429688
1	100	326.475311	0.796875
2	200	152.008636	0.867188
3	300	71.582230	0.898438
4	400	76.959229	0.859375
5	500	64.718910	0.789062
6	600	51.283031	0.851562
7	700	60.275085	0.867188
8	800	55.386650	0.882812
9	900	67.117737	0.851562
10	1000	52.590355	0.820312

In [17]:
```
plt.scatter(res_m1['step'], res_m1['loss'])
plt.axis([0,1000,0,1000])
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.title('Funcion de Costo')
```

Out[17]: Text(0.5,1,u'Funcion de Costo')



In [20]:
```
plt.scatter(res_m1['step'], res_m1['acc'])
plt.axis([0,1000,0.8,1])
plt.xlabel('Steps')
```

```
plt.ylabel('Accuracy')
plt.title('Precision del Modelo')
```

Out[20]: Text(0.5,1,u'Precision del Modelo')