IST597_week5

September 24, 2019

1 IST597:- Multi-Layer Perceptron

1.1 Load the libraries

In [36]:
```
import os
import numpy as np
import time
import tensorflow as tf
import tensorflow.contrib.eager as tfe
tf.enable_eager_execution()
tf.executing_eagerly()
```

Out[36]: True

In [37]:
```
tfe.num_gpus()
```

Out[37]: 1

#Get number of Gpu's in the system or else you can also use Nvidia-smi in command prompt

1.2 Generate random data

In [0]:
```
size_input = 32
size_hidden = 128
size_output = 1
number_of_train_examples = 1000
number_of_test_examples = 300
```

In [0]:
```
X_train = np.random.randn(number_of_train_examples , size_input)
y_train = np.random.randn(number_of_train_examples)
X_test = np.random.randn(number_of_test_examples, size_input)
y_test = np.random.randn(number_of_test_examples)
```

In [0]:
```
# Split dataset into batches
#train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).batch(16)
test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(4)
```

1.3 Build MLP using Eager Execution

```
In [0]: # Define class to build mlp model
        class MLP(object):
          def __init__(self, size_input, size_hidden, size_output, device=None):
            """
            size_input: int, size of input layer
            size_hidden: int, size of hidden layer
            size_output: int, size of output layer
            device: str or None, either 'cpu' or 'gpu' or None. If None, the device to be used u
            """
            self.size_input, self.size_hidden, self.size_output, self.device =\
            size_input, size_hidden, size_output, device

            # Initialize weights between input layer and hidden layer
            self.W1 = tfe.Variable(tf.random_normal([self.size_input, self.size_hidden]))
            # Initialize biases for hidden layer
            self.b1 = tfe.Variable(tf.random_normal([1, self.size_hidden]))
             # Initialize weights between hidden layer and output layer
            self.W2 = tfe.Variable(tf.random_normal([self.size_hidden, self.size_output]))
            # Initialize biases for output layer
            self.b2 = tfe.Variable(tf.random_normal([1, self.size_output]))

            # Define variables to be updated during backpropagation
            self.variables = [self.W1, self.W2, self.b1, self.b2]

          def forward(self, X):
            """
            forward pass
            X: Tensor, inputs
            """
            if self.device is not None:
              with tf.device('gpu:0' if self.device=='gpu' else 'cpu'):
                self.y = self.compute_output(X)
            else:
              self.y = self.compute_output(X)

            return self.y

          def loss(self, y_pred, y_true):
            '''
            y_pred - Tensor of shape (batch_size, size_output)
            y_true - Tensor of shape (batch_size, size_output)
            '''
            y_true_tf = tf.cast(tf.reshape(y_true, (-1, self.size_output)), dtype=tf.float32)
            y_pred_tf = tf.cast(y_pred, dtype=tf.float32)
            return tf.losses.mean_squared_error(y_true_tf, y_pred_tf)
```

```
    def backward(self, X_train, y_train):
      """
      backward pass
      """
      optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-4)
      with tf.GradientTape() as tape:
        predicted = self.forward(X_train)
        current_loss = self.loss(predicted, y_train)
      grads = tape.gradient(current_loss, self.variables)
      optimizer.apply_gradients(zip(grads, self.variables),
                                global_step=tf.train.get_or_create_global_step())


    def compute_output(self, X):
      """
      Custom method to obtain output tensor during forward pass
      """
      # Cast X to float32
      X_tf = tf.cast(X, dtype=tf.float32)
      #Remember to normalize your dataset before moving forward
      # Compute values in hidden layer
      what = tf.matmul(X_tf, self.W1) + self.b1
      hhat = tf.nn.relu(what)
      # Compute output
      output = tf.matmul(hhat, self.W2) + self.b2
      #Now consider two things , First look at inbuild loss functions if they work with so
      #Second add tf.Softmax(output) and then return this variable
      return output
```

1.4 Train Model

```
In [0]: # Set number of epochs
        NUM_EPOCHS = 10
```

```
In [43]: # Initialize model using CPU
         mlp_on_cpu = MLP(size_input, size_hidden, size_output, device='cpu')

         time_start = time.time()
         for epoch in range(NUM_EPOCHS):
           loss_total = tfe.Variable(0, dtype=tf.float32)
           train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(25, seed=ep
           for inputs, outputs in train_ds:
             preds = mlp_on_cpu.forward(inputs)
             loss_total = loss_total + mlp_on_cpu.loss(preds, outputs)
             mlp_on_cpu.backward(inputs, outputs)
           print('Number of Epoch = {} - Average MSE:= {:.4f}'.format(epoch + 1, loss_total.nump
         time_taken = time.time() - time_start
```

```
        print('\nTotal time taken (in seconds): {:.2f}'.format(time_taken))
        #For per epoch_time = Total_Time / Number_of_epochs
```

```
Number of Epoch = 1 - Average MSE:= 41.8356
Number of Epoch = 2 - Average MSE:= 17.2803
Number of Epoch = 3 - Average MSE:= 12.0315
Number of Epoch = 4 - Average MSE:= 10.0232
Number of Epoch = 5 - Average MSE:= 8.9033
Number of Epoch = 6 - Average MSE:= 8.0859
Number of Epoch = 7 - Average MSE:= 7.4388
Number of Epoch = 8 - Average MSE:= 6.8912
Number of Epoch = 9 - Average MSE:= 6.3923
Number of Epoch = 10 - Average MSE:= 5.9651

Total time taken (in seconds): 6.86
```

```
In [44]: # Initialize model using GPU
        mlp_on_gpu = MLP(size_input, size_hidden, size_output, device='gpu')

        time_start = time.time()
        for epoch in range(NUM_EPOCHS):
          loss_total = tfe.Variable(0, dtype=tf.float32)
          train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(25, seed=(e
          for inputs, outputs in train_ds:
            preds = mlp_on_gpu.forward(inputs)
            loss_total = loss_total + mlp_on_gpu.loss(preds, outputs)
            mlp_on_gpu.backward(inputs, outputs)
          print('Number of Epoch = {} - Average MSE:= {:.4f}'.format(epoch + 1, loss_total.nump
        time_taken = time.time() - time_start

        print('\nTotal time taken (in seconds): {:.2f}'.format(time_taken))
```

```
Number of Epoch = 1 - Average MSE:= 54.9124
Number of Epoch = 2 - Average MSE:= 24.0984
Number of Epoch = 3 - Average MSE:= 16.7387
Number of Epoch = 4 - Average MSE:= 13.9170
Number of Epoch = 5 - Average MSE:= 12.2832
Number of Epoch = 6 - Average MSE:= 11.1466
Number of Epoch = 7 - Average MSE:= 10.2290
Number of Epoch = 8 - Average MSE:= 9.4253
Number of Epoch = 9 - Average MSE:= 8.7257
Number of Epoch = 10 - Average MSE:= 8.1264

Total time taken (in seconds): 5.51
```

```
In [45]: #Default mode
        mlp_on_default = MLP(size_input, size_hidden, size_output)
```

```
        time_start = time.time()
        for epoch in range(NUM_EPOCHS):
          loss_total = tfe.Variable(0, dtype=tf.float32)
          train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(25, seed=(e
          for inputs, outputs in train_ds:
            preds = mlp_on_default.forward(inputs)
            loss_total = loss_total + mlp_on_default.loss(preds, outputs)
            mlp_on_default.backward(inputs, outputs)
          print('Epoch {} - Average MSE: {:.4f}'.format(epoch + 1, loss_total.numpy() / X_train
        time_taken = time.time() - time_start

        print('\nTotal time taken for training (secoonds): {:.2f}'.format(time_taken))
```

```
Epoch 1 - Average MSE: 41.2676
Epoch 2 - Average MSE: 19.7190
Epoch 3 - Average MSE: 14.6194
Epoch 4 - Average MSE: 12.3656
Epoch 5 - Average MSE: 10.9466
Epoch 6 - Average MSE: 9.9415
Epoch 7 - Average MSE: 9.1411
Epoch 8 - Average MSE: 8.4033
Epoch 9 - Average MSE: 7.7717
Epoch 10 - Average MSE: 7.2243

Total time taken for training (secoonds): 5.46
```

1.5 One Step Inference

```
In [46]: test_loss_total = tfe.Variable(0, dtype=tf.float32)
         for inputs, outputs in test_ds:
           preds = mlp_on_default.forward(inputs)
           test_loss_total = test_loss_total + mlp_on_default.loss(preds, outputs)
         print('Test MSE: {:.4f}'.format(test_loss_total.numpy() / X_train.shape[0]))
```

```
Test MSE: 16.3869
```