IST597_mlp

September 18, 2019

#IST597- softmax model(First step towards building your neural network)

```
In [0]: import tensorflow as tf
        import tensorflow.contrib.eager as tfe
        from tensorflow.examples.tutorials.mnist import input_data
```

Load tensorflow and mnist data_loader

```
In [0]: tfe.enable_eager_execution()
```

Check whether or not you're working in eager execution

```
In [0]: W = tf.get_variable(name="W", shape=(784, 10))
        b = tf.get_variable(name="b", shape=(10, ))
```

Create your tensorflow variables Create Weight and Biases

```
In [0]: def softmax_model(image_batch):
            model_output = tf.nn.softmax(tf.matmul(image_batch, W) + b)
            return model_output
```

We have created the softmax_model output = F(X.W+b) where X is input , W is weight and b is biases. F is non-linear function , softmax in our case

```
In [0]: def cross_entropy(model_output, label_batch):
            loss = tf.reduce_mean(
                -tf.reduce_sum(label_batch * tf.log(model_output),
                reduction_indices=[1]))
            return loss
```

Define your loss: In this case we would be using cross-entropy(NLL) Cross-entropy loss:- log loss is responsible for measuring the performance of a model which gives value between 0 and 1. A perfect model would have a log loss of 0.

```
In [0]: @tfe.implicit_value_and_gradients
        def cal_gradient(image_batch, label_batch):
            return cross_entropy(softmax_model(image_batch), label_batch)
```

This would returns a function which differentiates loss function with respect to variables.

In [33]:
```
lr=0.01
batch_size=64
epoch_n=10
data = input_data.read_data_sets("data/MNIST_data/", one_hot=True)
train_ds = tf.data.Dataset.from_tensor_slices((data.train.images, data.train.labels)).m
           .shuffle(buffer_size=1000)\
           .batch(batch_size=64)\


optimizer = tf.train.GradientDescentOptimizer(lr)

for epoch in range(epoch_n):
  for step, (image_batch, label_batch) in enumerate(tfe.Iterator(train_ds)):
      loss, grads_and_vars = cal_gradient(image_batch, label_batch)
      optimizer.apply_gradients(grads_and_vars)
      if(step%100 == 0):
        print("step: {}  loss: {}".format(step, loss.numpy()))

model_test_output = softmax_model(data.test.images)
model_test_label = data.test.labels
correct_prediction = tf.equal(tf.argmax(model_test_output, 1), tf.argmax(model_test_lab
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

print("test accuracy = {}".format(accuracy.numpy()))
```

```
Extracting data/MNIST_data/train-images-idx3-ubyte.gz
Extracting data/MNIST_data/train-labels-idx1-ubyte.gz
Extracting data/MNIST_data/t10k-images-idx3-ubyte.gz
Extracting data/MNIST_data/t10k-labels-idx1-ubyte.gz
step: 0  loss: 2.4309253692626953
step: 100  loss: 1.6556241512298584
step: 200  loss: 1.2155452966690063
step: 300  loss: 1.0475245714187622
```

```
step: 400  loss: 1.0862095355987549
step: 500  loss: 0.7784752249717712
step: 600  loss: 0.8664791584014893
step: 700  loss: 0.7498409748077393
step: 800  loss: 0.698955774307251
step: 0  loss: 0.6215534210205078
step: 100  loss: 0.643366813659668
step: 200  loss: 0.5203976631164551
step: 300  loss: 0.6068369150161743
step: 400  loss: 0.7616375088691711
step: 500  loss: 0.5042001008987427
step: 600  loss: 0.6564253568649292
step: 700  loss: 0.5584914088249207
step: 800  loss: 0.5750884413719177
step: 0  loss: 0.47606679797172546
step: 100  loss: 0.5176450610160828
step: 200  loss: 0.4088672995567322
step: 300  loss: 0.5149017572402954
step: 400  loss: 0.6724463105201721
step: 500  loss: 0.4298582673072815
step: 600  loss: 0.584412693977356
step: 700  loss: 0.4898722171783447
step: 800  loss: 0.5216456651687622
step: 0  loss: 0.4172459840774536
step: 100  loss: 0.46674051880836487
step: 200  loss: 0.3576718270778656
step: 300  loss: 0.47118520736694336
step: 400  loss: 0.6281046271324158
step: 500  loss: 0.39349788427352905
step: 600  loss: 0.5446990728378296
step: 700  loss: 0.45289987325668335
step: 800  loss: 0.48751479387283325
step: 0  loss: 0.38431769609451294
step: 100  loss: 0.43784481287002563
step: 200  loss: 0.3266371488571167
step: 300  loss: 0.44390714168548584
step: 400  loss: 0.6007729768753052
step: 500  loss: 0.3711305856704712
step: 600  loss: 0.5183553099632263
step: 700  loss: 0.42935237288475037
step: 800  loss: 0.4623306393623352
step: 0  loss: 0.36290255188941956
step: 100  loss: 0.4184116721153259
step: 200  loss: 0.30524325370788574
step: 300  loss: 0.42441099882125854
step: 400  loss: 0.581855058670044
step: 500  loss: 0.3556104302406311
step: 600  loss: 0.4991150200366974
```

```
step: 700  loss: 0.4128991961479187
step: 800  loss: 0.4423584043979645
step: 0  loss: 0.34770071506500244
step: 100  loss: 0.4039914906024933
step: 200  loss: 0.28935688734054565
step: 300  loss: 0.40934592485427856
step: 400  loss: 0.5677635669708252
step: 500  loss: 0.3440224528312683
step: 600  loss: 0.4842134714126587
step: 700  loss: 0.4006926417350769
step: 800  loss: 0.4258308708667755
step: 0  loss: 0.33626943826675415
step: 100  loss: 0.39260417222976685
step: 200  loss: 0.27697357535362244
step: 300  loss: 0.39712393283843994
step: 400  loss: 0.5567216873168945
step: 500  loss: 0.3349366784095764
step: 600  loss: 0.4722058176994324
step: 700  loss: 0.3912419080734253
step: 800  loss: 0.4117637276649475
step: 0  loss: 0.3273158669471741
step: 100  loss: 0.38322532176971436
step: 200  loss: 0.26698529720306396
step: 300  loss: 0.3868834376335144
step: 400  loss: 0.547744631767273
step: 500  loss: 0.32756030559539795
step: 600  loss: 0.4622482657432556
step: 700  loss: 0.38368332386016846
step: 800  loss: 0.3995473384857178
step: 0  loss: 0.32008713483810425
step: 100  loss: 0.37526506185531616
step: 200  loss: 0.25872135162353516
step: 300  loss: 0.3781081438064575
step: 400  loss: 0.5402401685714722
step: 500  loss: 0.3214142322540283
step: 600  loss: 0.4538081884384155
step: 700  loss: 0.37748056650161743
step: 800  loss: 0.3887752890586853
test accuracy = 0.90829998254776
```