Epochs

An epoch is a single forward and backward pass of the whole dataset. This is used to increase the accuracy of the model without requiring more data. This section will cover epochs in TensorFlow and how to choose the right number of epochs.

The following TensorFlow code trains a model using 10 epochs.

```
from tensorflow.examples.tutorials.mnist import input_data
import tensorflow as tf
import numpy as np
from helper import batches  # Helper function created in Mini-batching section


def print_epoch_stats(epoch_i, sess, last_features, last_labels):
    """
    Print cost and validation accuracy of an epoch
    """
    current_cost = sess.run(
        cost,
        feed_dict={features: last_features, labels: last_labels})
    valid_accuracy = sess.run(
        accuracy,
        feed_dict={features: valid_features, labels: valid_labels})
    print('Epoch: {:<4} - Cost: {:<8.3} Valid Accuracy: {:<5.3}'.format(
        epoch_i,
        current_cost,
        valid_accuracy))

n_input = 784  # MNIST data input (img shape: 28*28)
n_classes = 10  # MNIST total classes (0-9 digits)

# Import MNIST data
mnist = input_data.read_data_sets('/datasets/ud730/mnist', one_hot=True)

# The features are already scaled and the data is shuffled
train_features = mnist.train.images
valid_features = mnist.validation.images
test_features = mnist.test.images

train_labels = mnist.train.labels.astype(np.float32)
valid_labels = mnist.validation.labels.astype(np.float32)
test_labels = mnist.test.labels.astype(np.float32)

# Features and Labels
```

```
features = tf.placeholder(tf.float32, [None, n_input])
labels = tf.placeholder(tf.float32, [None, n_classes])

# Weights & bias
weights = tf.Variable(tf.random_normal([n_input, n_classes]))
bias = tf.Variable(tf.random_normal([n_classes]))

# Logits - xW + b
logits = tf.add(tf.matmul(features, weights), bias)

# Define loss and optimizer
learning_rate = tf.placeholder(tf.float32)
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, la
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).mini

# Calculate accuracy
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(labels, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

init = tf.global_variables_initializer()

batch_size = 128
epochs = 10
learn_rate = 0.001

train_batches = batches(batch_size, train_features, train_labels)

with tf.Session() as sess:
    sess.run(init)

    # Training cycle
    for epoch_i in range(epochs):

        # Loop over all batches
        for batch_features, batch_labels in train_batches:
            train_feed_dict = {
                features: batch_features,
                labels: batch_labels,
                learning_rate: learn_rate}
            sess.run(optimizer, feed_dict=train_feed_dict)

        # Print cost and validation accuracy of an epoch
        print_epoch_stats(epoch_i, sess, batch_features, batch_labels)

    # Calculate accuracy for test dataset
    test_accuracy = sess.run(
        accuracy,
        feed_dict={features: test_features, labels: test_labels})
```

```
print('Test Accuracy: {}'.format(test_accuracy))
```

Running the code will output the following:

```
Epoch: 0    - Cost: 11.0     Valid Accuracy: 0.204
Epoch: 1    - Cost: 9.95     Valid Accuracy: 0.229
Epoch: 2    - Cost: 9.18     Valid Accuracy: 0.246
Epoch: 3    - Cost: 8.59     Valid Accuracy: 0.264
Epoch: 4    - Cost: 8.13     Valid Accuracy: 0.283
Epoch: 5    - Cost: 7.77     Valid Accuracy: 0.301
Epoch: 6    - Cost: 7.47     Valid Accuracy: 0.316
Epoch: 7    - Cost: 7.2      Valid Accuracy: 0.328
Epoch: 8    - Cost: 6.96     Valid Accuracy: 0.342
Epoch: 9    - Cost: 6.73     Valid Accuracy: 0.36
Test Accuracy: 0.3801000118255615
```

Each epoch attempts to move to a lower cost, leading to better accuracy.

This model continues to improve accuracy up to Epoch 9. Let's increase the number of epochs to 100.

```
...
Epoch: 79   - Cost: 0.111    Valid Accuracy: 0.86
Epoch: 80   - Cost: 0.11     Valid Accuracy: 0.869
Epoch: 81   - Cost: 0.109    Valid Accuracy: 0.869
....
Epoch: 85   - Cost: 0.107    Valid Accuracy: 0.869
Epoch: 86   - Cost: 0.107    Valid Accuracy: 0.869
Epoch: 87   - Cost: 0.106    Valid Accuracy: 0.869
Epoch: 88   - Cost: 0.106    Valid Accuracy: 0.869
Epoch: 89   - Cost: 0.105    Valid Accuracy: 0.869
Epoch: 90   - Cost: 0.105    Valid Accuracy: 0.869
Epoch: 91   - Cost: 0.104    Valid Accuracy: 0.869
Epoch: 92   - Cost: 0.103    Valid Accuracy: 0.869
Epoch: 93   - Cost: 0.103    Valid Accuracy: 0.869
Epoch: 94   - Cost: 0.102    Valid Accuracy: 0.869
Epoch: 95   - Cost: 0.102    Valid Accuracy: 0.869
Epoch: 96   - Cost: 0.101    Valid Accuracy: 0.869
Epoch: 97   - Cost: 0.101    Valid Accuracy: 0.869
Epoch: 98   - Cost: 0.1      Valid Accuracy: 0.869
Epoch: 99   - Cost: 0.1      Valid Accuracy: 0.869
Test Accuracy: 0.8696000006198883
```

From looking at the output above, you can see the model doesn't increase the validation accuracy after epoch 80. Let's see what happens when we increase the learning rate.

learn_rate = 0.1

```
Epoch: 76   - Cost: 0.214    Valid Accuracy: 0.752
Epoch: 77   - Cost: 0.21     Valid Accuracy: 0.756
Epoch: 78   - Cost: 0.21     Valid Accuracy: 0.756
...
Epoch: 85   - Cost: 0.207    Valid Accuracy: 0.756
Epoch: 86   - Cost: 0.209    Valid Accuracy: 0.756
Epoch: 87   - Cost: 0.205    Valid Accuracy: 0.756
Epoch: 88   - Cost: 0.208    Valid Accuracy: 0.756
Epoch: 89   - Cost: 0.205    Valid Accuracy: 0.756
Epoch: 90   - Cost: 0.202    Valid Accuracy: 0.756
Epoch: 91   - Cost: 0.207    Valid Accuracy: 0.756
Epoch: 92   - Cost: 0.204    Valid Accuracy: 0.756
Epoch: 93   - Cost: 0.206    Valid Accuracy: 0.756
Epoch: 94   - Cost: 0.202    Valid Accuracy: 0.756
Epoch: 95   - Cost: 0.2974   Valid Accuracy: 0.756
Epoch: 96   - Cost: 0.202    Valid Accuracy: 0.756
Epoch: 97   - Cost: 0.2996   Valid Accuracy: 0.756
Epoch: 98   - Cost: 0.203    Valid Accuracy: 0.756
Epoch: 99   - Cost: 0.2987   Valid Accuracy: 0.756
Test Accuracy: 0.7556000053882599
```

Looks like the learning rate was increased too much. The final accuracy was lower, and it stopped improving earlier. Let's stick with the previous learning rate, but change the number of epochs to 80.

```
Epoch: 65   - Cost: 0.122    Valid Accuracy: 0.868
Epoch: 66   - Cost: 0.121    Valid Accuracy: 0.868
Epoch: 67   - Cost: 0.12     Valid Accuracy: 0.868
Epoch: 68   - Cost: 0.119    Valid Accuracy: 0.868
Epoch: 69   - Cost: 0.118    Valid Accuracy: 0.868
Epoch: 70   - Cost: 0.118    Valid Accuracy: 0.868
Epoch: 71   - Cost: 0.117    Valid Accuracy: 0.868
Epoch: 72   - Cost: 0.116    Valid Accuracy: 0.868
Epoch: 73   - Cost: 0.115    Valid Accuracy: 0.868
Epoch: 74   - Cost: 0.115    Valid Accuracy: 0.868
Epoch: 75   - Cost: 0.114    Valid Accuracy: 0.868
Epoch: 76   - Cost: 0.113    Valid Accuracy: 0.868
```

```
Epoch: 77   - Cost: 0.113    Valid Accuracy: 0.868
Epoch: 78   - Cost: 0.112    Valid Accuracy: 0.868
Epoch: 79   - Cost: 0.111    Valid Accuracy: 0.868
Epoch: 80   - Cost: 0.111    Valid Accuracy: 0.869
Test Accuracy: 0.86909999418258667
```

The accuracy only reached 0.86, but that could be because the learning rate was too high. Lowering the learning rate would require more epochs, but could ultimately achieve better accuracy.

In the upcoming TensorFLow Lab, you'll get the opportunity to choose your own learning rate, epoch count, and batch size to improve the model's accuracy.