Weights and Bias in TensorFlow

The goal of training a neural network is to modify weights and biases to best predict the labels. In order to use weights and bias, you'll need a Tensor that can be modified. This leaves out `tf.placeholder()` and `tf.constant()`, since those Tensors can't be modified. This is where `tf.Variable` class comes in.

tf.Variable()

```
x = tf.Variable(5)
```

The `tf.Variable` class creates a tensor with an initial value that can be modified, much like a normal Python variable. This tensor stores its state in the session, so you must initialize the state of the tensor manually. You'll use the `tf.global_variables_initializer()` function to initialize the state of all the Variable tensors.

Initialization

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
```

The `tf.global_variables_initializer()` call returns an operation that will initialize all TensorFlow variables from the graph. You call the operation using a session to initialize all the variables as shown above. Using the `tf.Variable` class allows us to change the weights and bias, but an initial value needs to be chosen.

Initializing the weights with random numbers from a normal distribution is good practice. Randomizing the weights helps the model from becoming stuck in the same place every time you train it. You'll learn more about this in the next lesson, when you study gradient descent.

Similarly, choosing weights from a normal distribution prevents any one weight from overwhelming other weights. You'll use the `tf.truncated_normal()`function to generate random numbers from a normal distribution.

tf.truncated_normal()

```
n_features = 120
n_labels = 5
weights = tf.Variable(tf.truncated_normal((n_features, n_labels)))
```

The `tf.truncated_normal()` function returns a tensor with random values from a normal distribution whose magnitude is no more than 2 standard deviations from the mean.

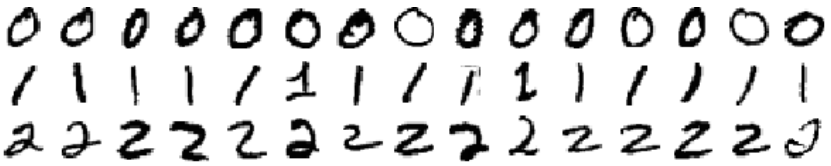
Since the weights are already helping prevent the model from getting stuck, you don't need to randomize the bias. Let's use the simplest solution, setting the bias to 0.

tf.zeros()

```
n_labels = 5
bias = tf.Variable(tf.zeros(n_labels))
```

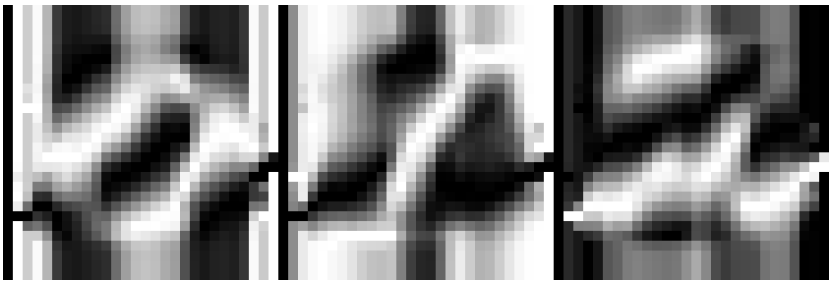
The `tf.zeros()` function returns a tensor with all zeros.

Linear Classifier Quiz



Subset of MNIST dataset.

You'll be classifying the handwritten numbers 0, 1, and 2 from the MNIST dataset using TensorFlow. The above is a small sample of the data you'll be training on. Notice how some of the 1s are written with a serif at the top and at different angles. The similarities and differences will play a part in shaping the weights of the model.



Left: Weights for labeling 0. Middle: Weights for labeling 1. Right: Weights for labeling 2.

The images above are trained weights for each label (`0`, `1`, and `2`). The weights display the unique properties of each digit they have found. Complete this quiz to train your own weights using the MNIST dataset.

Instructions

1. Open quiz.py.
    1. Implement `get_weights` to return a **`tf.Variable`** of weights
    2. Implement `get_biases` to return a **`tf.Variable`** of biases
    3. Implement `xW + b` in the `linear` function
2. Open sandbox.py
    1. Initialize all weights

Since `xW` in `xW + b` is matrix multiplication, you have to use the **`tf.matmul()`** function instead of **`tf.multiply()`**. Don't forget that order matters in matrix multiplication, so `tf.matmul(a,b)` is not the same as `tf.matmul(b,a)`.