A. Running the model

In this chapter and the next, you will be running your model on input data, using a `tf.Session` object and the `tf.placeholder` variables from the previous chapters.

The `tf.Session` object has an extremely important function called `run`. All the code written in the previous chapters was to build the computation graph of the neural network, i.e. its layers and operations. However, we can only train or evaluate the model on real input data using `run`. The function takes in a single required argument and a few keyword arguments.

B. Using `run`

The required argument is normally either a single tensor/operation or a list/tuple of tensors and operations. Calling `run` on a tensor returns the value of that tensor after executing our computation graph. The output of `run` with a tensor input is a NumPy array.

The code below shows usages of `run` on `tf.constant` tensors.

```
t = tf.constant([1, 2, 3])
sess = tf.Session()
arr = sess.run(t)
print('{}\n'.format(repr(arr)))

t2 = tf.constant(4)
tup = sess.run((t, t2))
print('{}\n'.format(repr(tup)))
```

RUN SAVE RESET

Close

Output 3.794s

```
array([1, 2, 3], dtype=int32)

(array([1, 2, 3], dtype=int32), 4)
```

Of the keyword arguments for `run`, the important one for most applications is `feed_dict`. The `feed_dict` is a python dictionary. Each key is a *tensor* from the model's computation graph. The key's value can be a Python scalar, list, or NumPy array.

We use `feed_dict` to pass values into certain tensors in the computation graph. In the code below, we pass in a value for `inputs`, which is a `tf.placeholder` object.

```
inputs = tf.placeholder(tf.float32, shape=(None, 2))
feed_dict = {
  inputs: [[1.1, -0.3],
           [0.2, 0.1]]
}
sess = tf.Session()
arr = sess.run(inputs, feed_dict=feed_dict)
print('{}\n'.format(repr(arr)))
```

RUN SAVE RESET

Close

Output 3.432s

```
array([[ 1.1, -0.3],
       [ 0.2,  0.1]], dtype=float32)
```

Each `tf.placeholder` object used in the model execution must be included as a key in the `feed_dict`, with the corresponding value's shape and type matching the placeholder's.

C. Initializing variables

When we call `run`, every tensor in the model's computation graph must either already have a value or must be fed in a value through `feed_dict`. However, when we start training from scratch, none of our variables (e.g. weights) have values yet. We need to initialize all the variables using `tf.global_variables_initializer`. This returns an operation that, when used as the required argument in `run`, initializes all the variables in the model.

In the code below, the variables that are initialized are part of `tf.layers.dense`. The variable initialization process is defined internally by the function. In this case, the variables are initialized in a way that results in zero logits.

```
inputs = tf.placeholder(tf.float32, shape=(None, 2))
feed_dict = {
    inputs: [[1.1, -0.3],
             [0.2, 0.1]]
}
logits = tf.layers.dense(inputs, 1, name='logits')
init_op = tf.global_variables_initializer()

sess = tf.Session()
sess.run(init_op) # variable initialization
arr = sess.run(logits, feed_dict=feed_dict)
print('{}\n'.format(repr(arr)))
```

RUN SAVE RESET

Close

Output 3.435s

```
array([[ 0.5483774 ],
       [-0.00287547]], dtype=float32)
```

D. Training logistics

The `num_steps` argument represents the number of iterations we use to train our model. Each iteration we train the model on a *batch* of data points. So `input_data` is essentially a large dataset divided into chunks (i.e. batches), and each iteration we train on a specific batch of points and their corresponding labels.

```
# predefined dataset
print('Input data:')
print('{}\n'.format(repr(input_data)))

print('Labels:')
print('{}\n'.format(repr(input_labels)))
```

RUN SAVE RESET Close

Output 0.663s

```
Input data:
array([[[ 1.2,  0.3],
        [ 0.1, -0.2],
        [-0.3,  0.3]],

       [[-2.1,  1.4],
        [-0.9,  0.6],
        [ 1.2,  2.2]],
```