TensorFlow ReLUs

TensorFlow provides the ReLU function as `tf.nn.relu()`, as shown below.

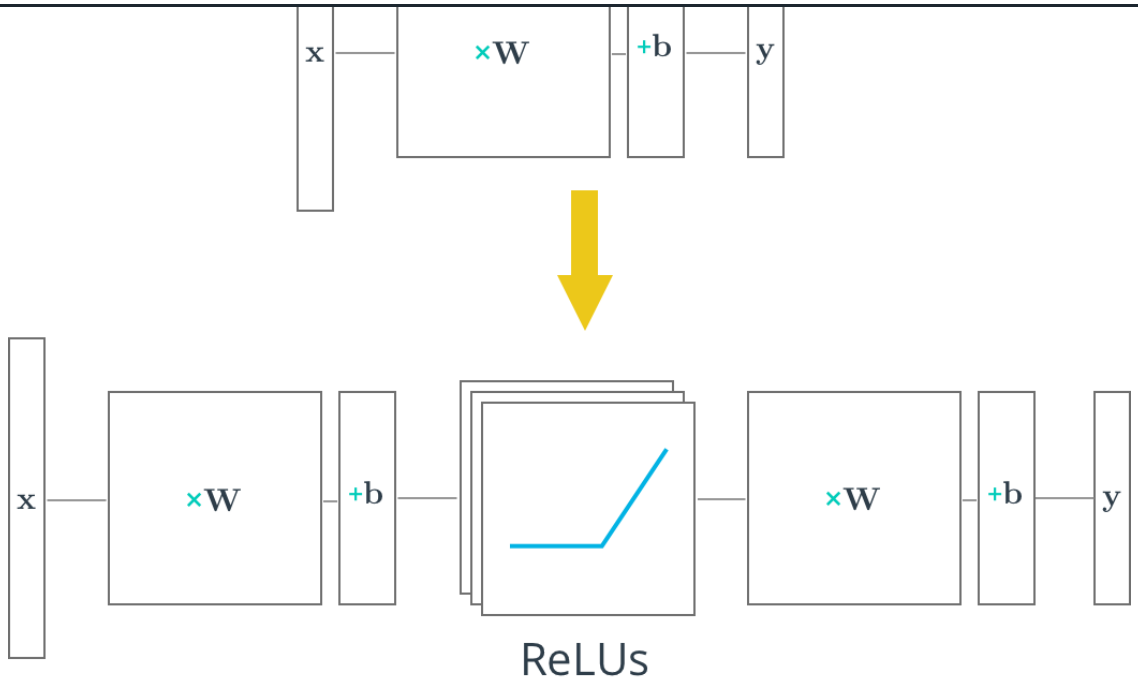
```
# Hidden Layer with ReLU activation function
hidden_layer = tf.add(tf.matmul(features, hidden_weights), hidden_biases
hidden_layer = tf.nn.relu(hidden_layer)

output = tf.add(tf.matmul(hidden_layer, output_weights), output_biases)
```

The above code applies the `tf.nn.relu()` function to the `hidden_layer`, effectively turning off any negative weights and acting like an on/off switch. Adding additional layers, like the `output` layer, after an activation function turns the model into a nonlinear function. This nonlinearity allows the network to solve more complex problems.

Quiz

Below you'll use the ReLU function to turn a linear single layer network into a non-linear multilayer network.

Quiz: TensorFlow ReLUs



quiz.py | solution.py

```
    [0.1, 0.2, 0.4],
    [0.4, 0.6, 0.6],
    [0.5, 0.9, 0.1],
    [0.8, 0.2, 0.8]]
out_weights = [
    [0.1, 0.6],
    [0.2, 0.1],
    [0.7, 0.9]]

# Weights and biases
weights = [
    tf.Variable(hidden_layer_weights),
    tf.Variable(out_weights)]
biases = [
    tf.Variable(tf.zeros(3)),
    tf.Variable(tf.zeros(2))]

# Input
features = tf.Variable([[1.0, 2.0, 3.0, 4.0], [-1.0, -2.0, -3.0, -4.0], [11.0,

# TODO: Create Model
hidden_layer = tf.add(tf.matmul(features,weights[0]),biases[0])
hidden_layer = tf.nn.relu(hidden_layer)
logits = tf.add(tf.matmul(hidden_layer,weights[1]), biases[1])

# TODO: Print session results
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    print(sess.run(logits))

```

Quiz: TensorFlow ReLUs

```
[ 24.01000214  38.23999786]]
```

RESET QUIZ TEST RUN SUBMIT ANSWER

NEXT