

Deep Neural Networks using TensorBoard

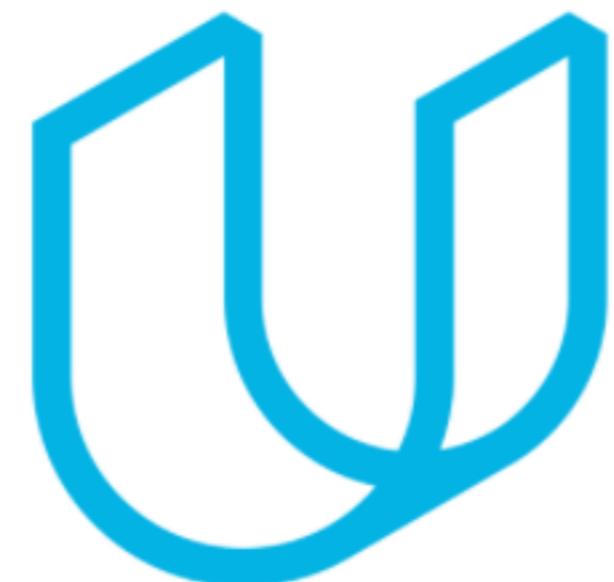
github.com/PythonWorkshop/tensorboard_demos

Luis Serrano

[@luis_likes_math
github.com/luisguiserrano](https://github.com/luisguiserrano)

Arpan Chakraborty

[@runoffthemill
github.com/napratin](https://github.com/napratin)



UDACITY

udacity.com/ai

udacity.com/ml



Getting Started

Note: This notebook is written in Python 3. You'll need Jupyter/iPython to run it.

Local installation

Fetch the repo: github.com/PythonWorkshop/tensorboard_demos

```
git clone git@github.com:PythonWorkshop/tensorboard_demos  
cd tensorboard_demos/
```

Option A: Conda install

```
conda env create  
source activate tensorflow  
jupyter notebook tensorboard_basics.ipynb
```

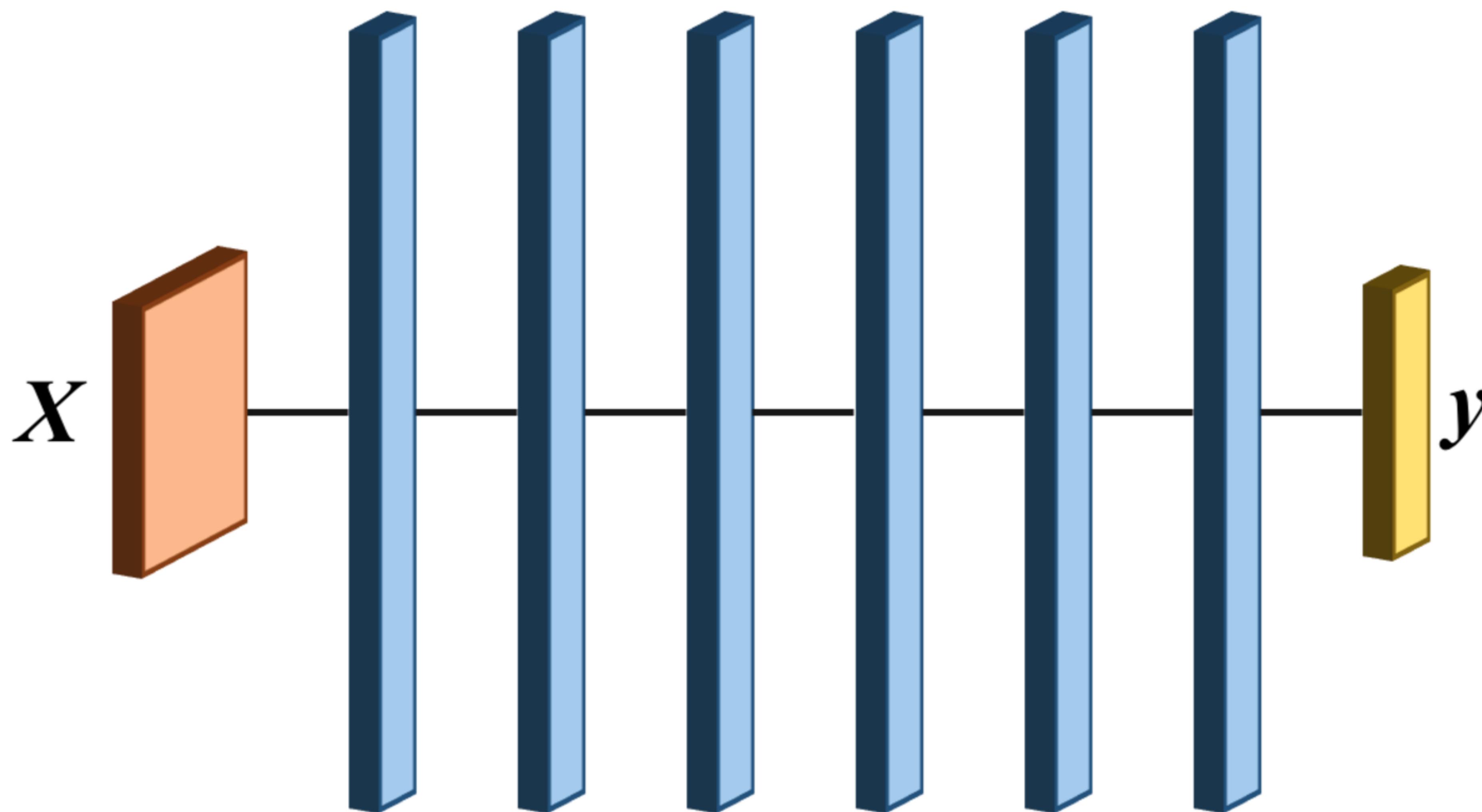
Option B: Pip install

```
pip3 install numpy matplotlib scikit-learn  
pip3 install --upgrade <binary URL for your system>  
jupyter notebook tensorboard_basics.ipynb
```

See [TensorFlow instructions](#) to pick the correct binary URL for your system.



Deep Neural Networks

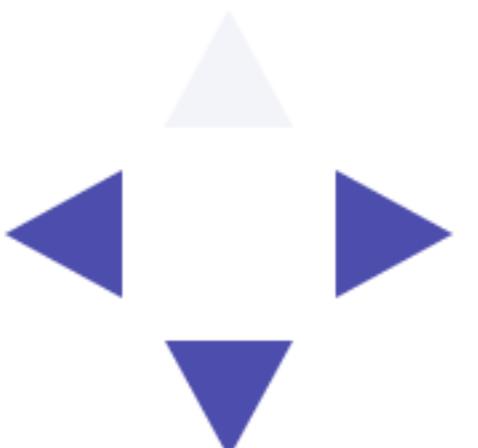


Generate Some Data to Classify

```
In [1]: import numpy as np

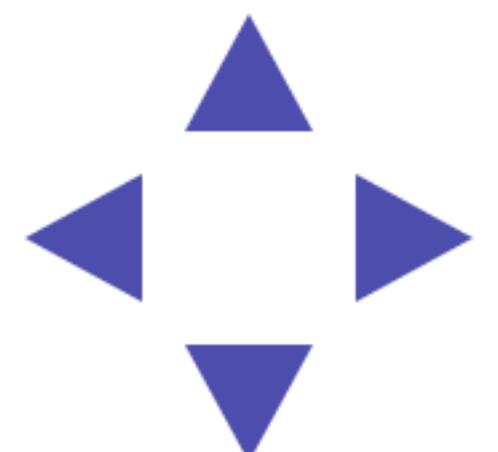
num_examples, num_features = (1000, 2) # dataset size
num_classes = 2 # binary classification task

X = np.random.random((num_examples, num_features))
y = np.int_(X[:, 0] * X[:, 0] + X[:, 1] >= 1).reshape(-1, 1)
```



```
In [2]: print("Features (X): {shape[0]}x{shape[1]}".format(shape=X.shape))
print(X[:10])
```

```
Features (X): 1000x2
[[ 0.63088627  0.64420886]
 [ 0.78599865  0.24681372]
 [ 0.96552681  0.99162588]
 [ 0.39100326  0.85880169]
 [ 0.87449078  0.74400594]
 [ 0.89062883  0.98642324]
 [ 0.30009226  0.16946355]
 [ 0.25510643  0.90216481]
 [ 0.88847416  0.60861312]
 [ 0.24249016  0.70518724]]
```



```
In [3]: print("Labels (y): {shape[0]}".format(shape=y.shape))
print(y[:10])
```

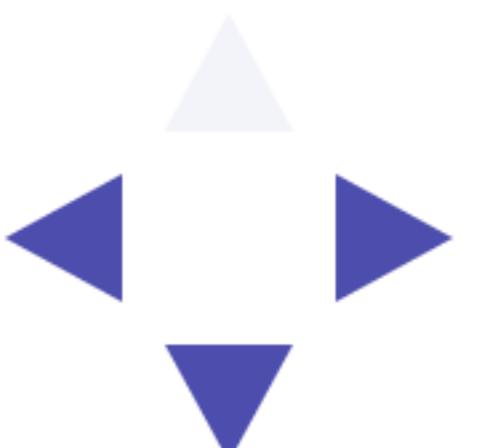
```
Labels (y): 1000
[[1]
 [0]
 [1]
 [1]
 [1]
 [1]
 [0]
 [0]
 [1]
 [0]]
```



Visualize Data

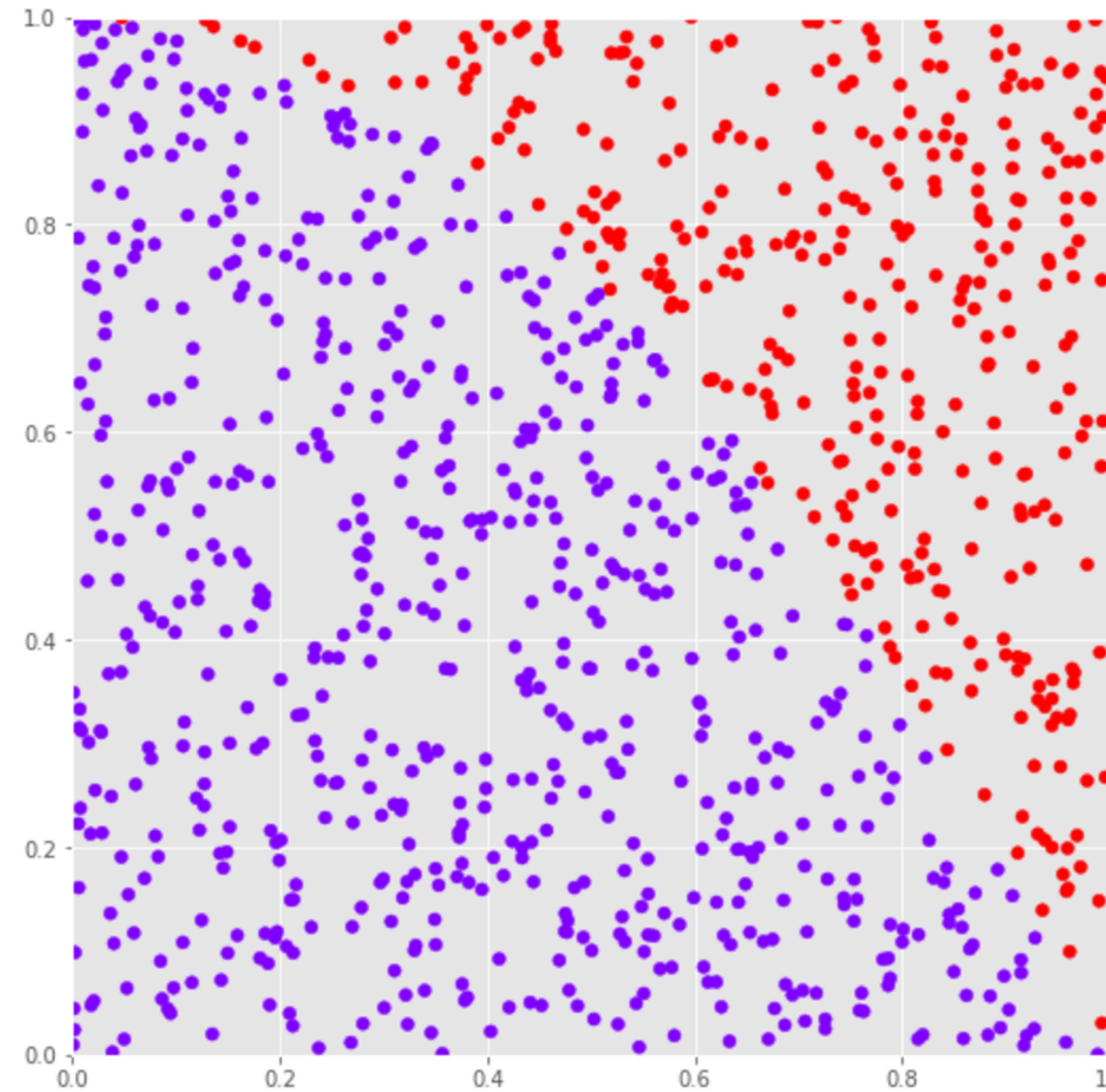
```
In [4]: import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')

def plot_data_2D(X, **kwargs):
    """Plot 2D data points in X = np.array([(x0, y0), (x1, y1), ...])."""
    fig, ax = plt.subplots(figsize=(9, 9))
    ax.scatter(X[:, 0], X[:, 1],
               s=35, cmap=plt.cm.get_cmap('rainbow', num_classes), **kwargs)
    ax.set_aspect('equal')
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    return fig, ax
```



```
In [5]: plot_data_2D(X, c=y)
```

```
Out[5]: (<matplotlib.figure.Figure at 0x102a1f9e8>,
<matplotlib.axes._subplots.AxesSubplot at 0x1031ceba8>)
```

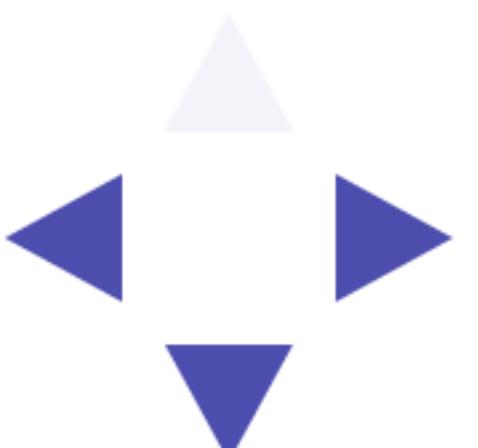


Prepare Data for Training and Testing

```
In [6]: # Split data into training and test sets
from sklearn.cross_validation import train_test_split

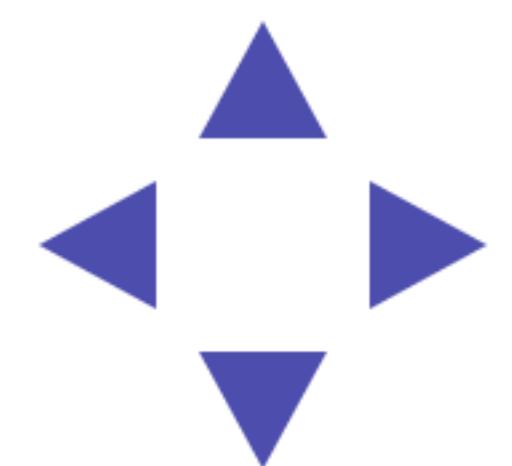
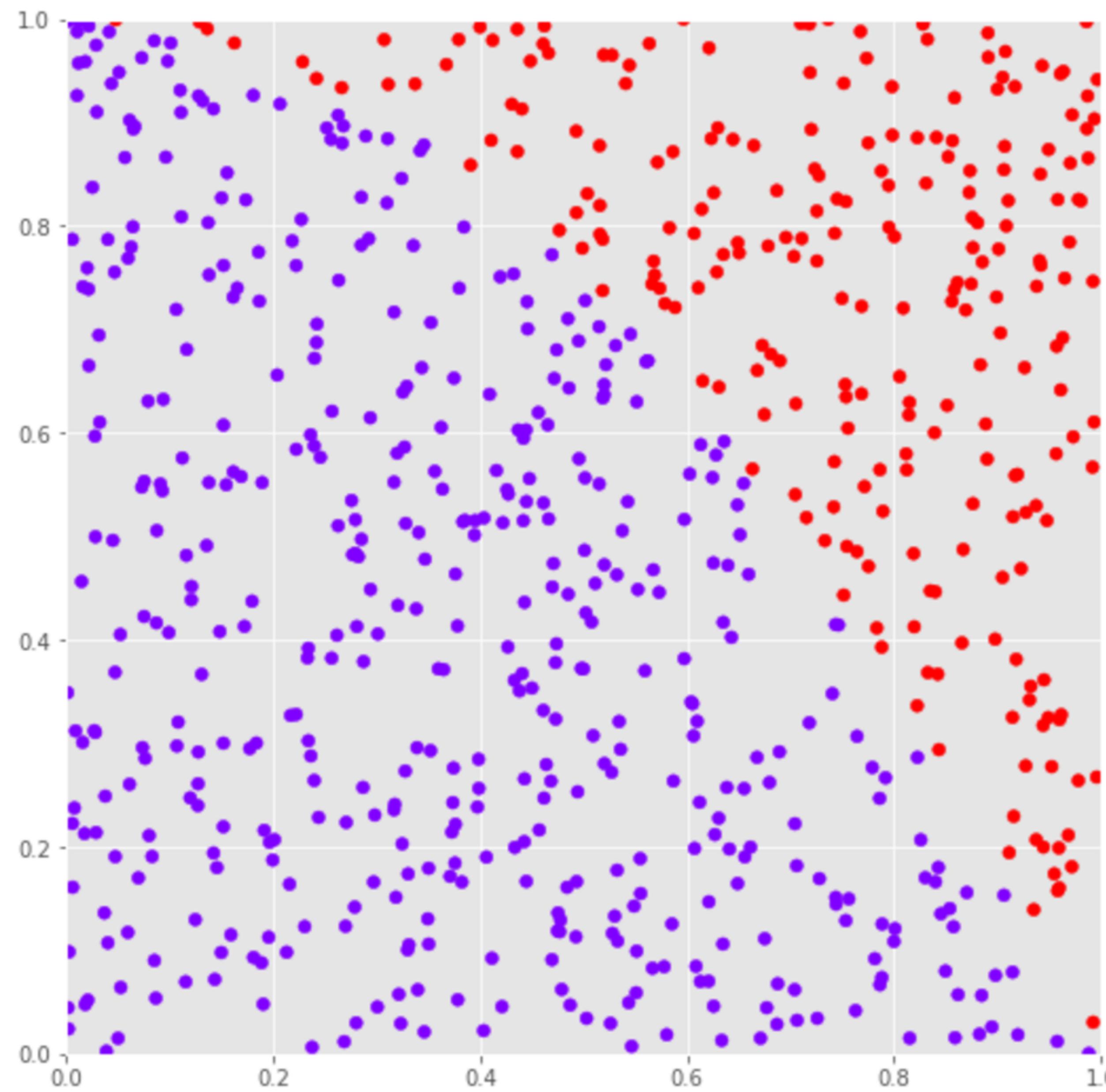
test_size = 300
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
print("Split dataset: {} training, {} test samples".format(len(X_train), len(X_test)))
```

Split dataset: 700 training, 300 test samples



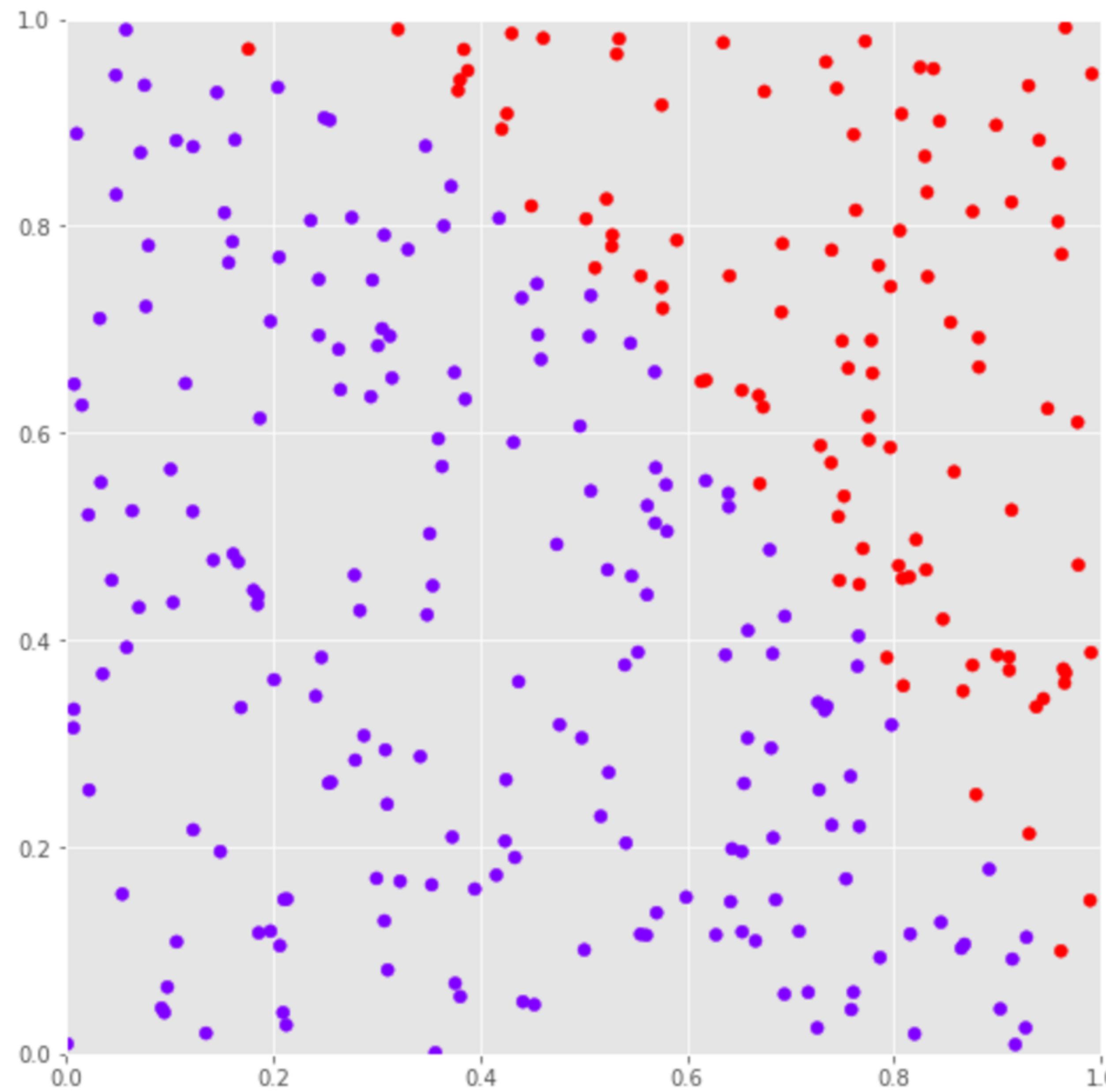
```
In [7]: # Plot training data  
plot_data_2D(X_train, c=y_train)
```

```
Out[7]: (<matplotlib.figure.Figure at 0x1040d5550>,  
<matplotlib.axes._subplots.AxesSubplot at 0x106235f28>)
```



```
In [8]: # Plot test data  
plot_data_2D(X_test, c=y_test)
```

```
Out[8]: (<matplotlib.figure.Figure at 0x106271470>,  
<matplotlib.axes._subplots.AxesSubplot at 0x10624bfd0>)
```



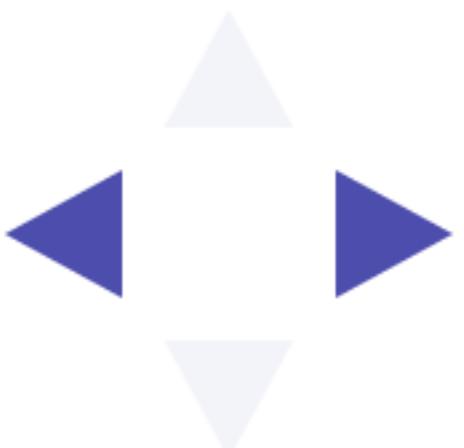
Build Computation Graph

This graph defines the structure of your Neural Network as well as information flow.

```
In [9]: import tensorflow as tf

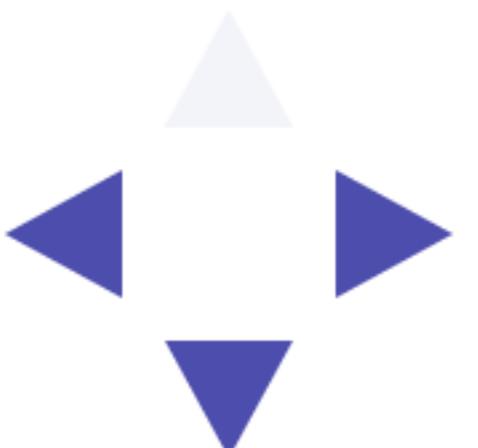
# Create a TensorFlow session
session = tf.Session()

# Placeholders for input features and ground truth labels
x_placeholder = tf.placeholder(tf.float32,
                               shape=(None, num_features),
                               name="X")
y_placeholder = tf.placeholder(tf.int64,
                               shape=(None, 1),
                               name="y")
```

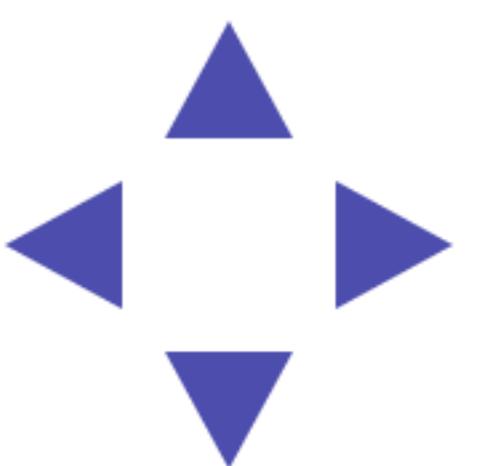
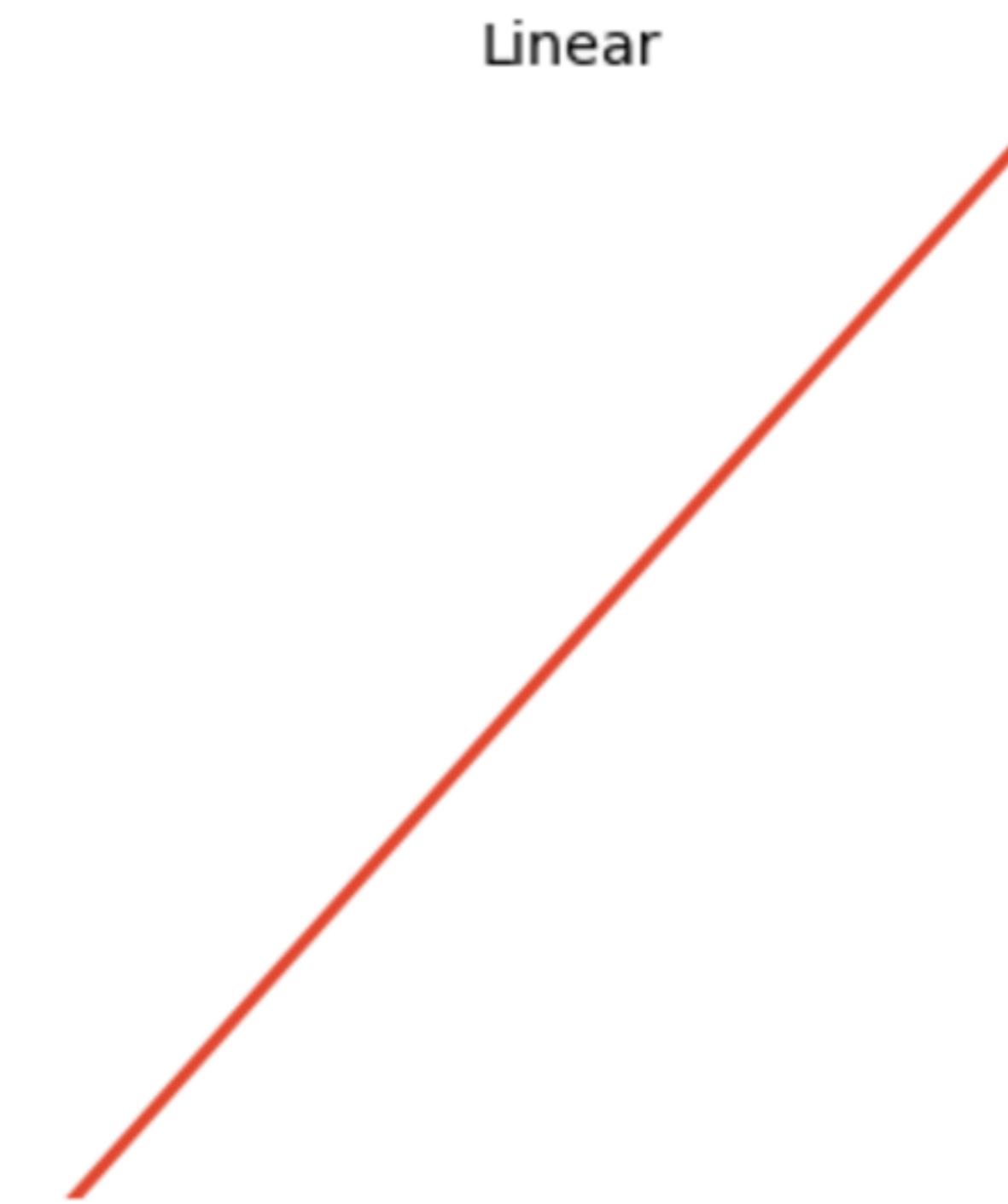


Define layer creation functions

```
In [11]: def linear_layer(input_tensor, num_units):
    """Linear activation layer: output = inputs * weights + biases"""
    num_inputs = input_tensor.get_shape()[1].value # inspect tensor size
    weights = tf.Variable(
        tf.truncated_normal([num_inputs, num_units], stddev=1.0),
        name='weights')
    biases = tf.Variable(
        tf.zeros([num_units]),
        name='biases')
    return tf.add(tf.matmul(input_tensor, weights, name='multiply'), biases,
                 name='add')
```

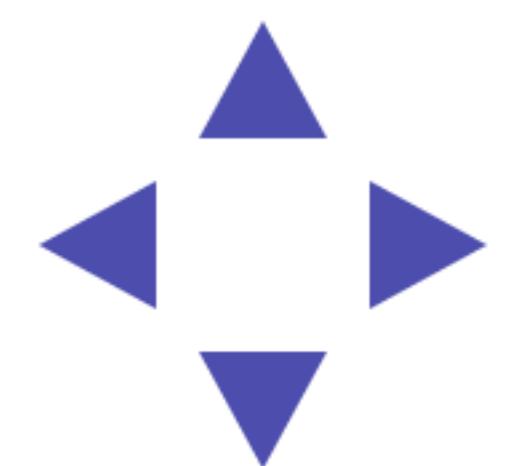
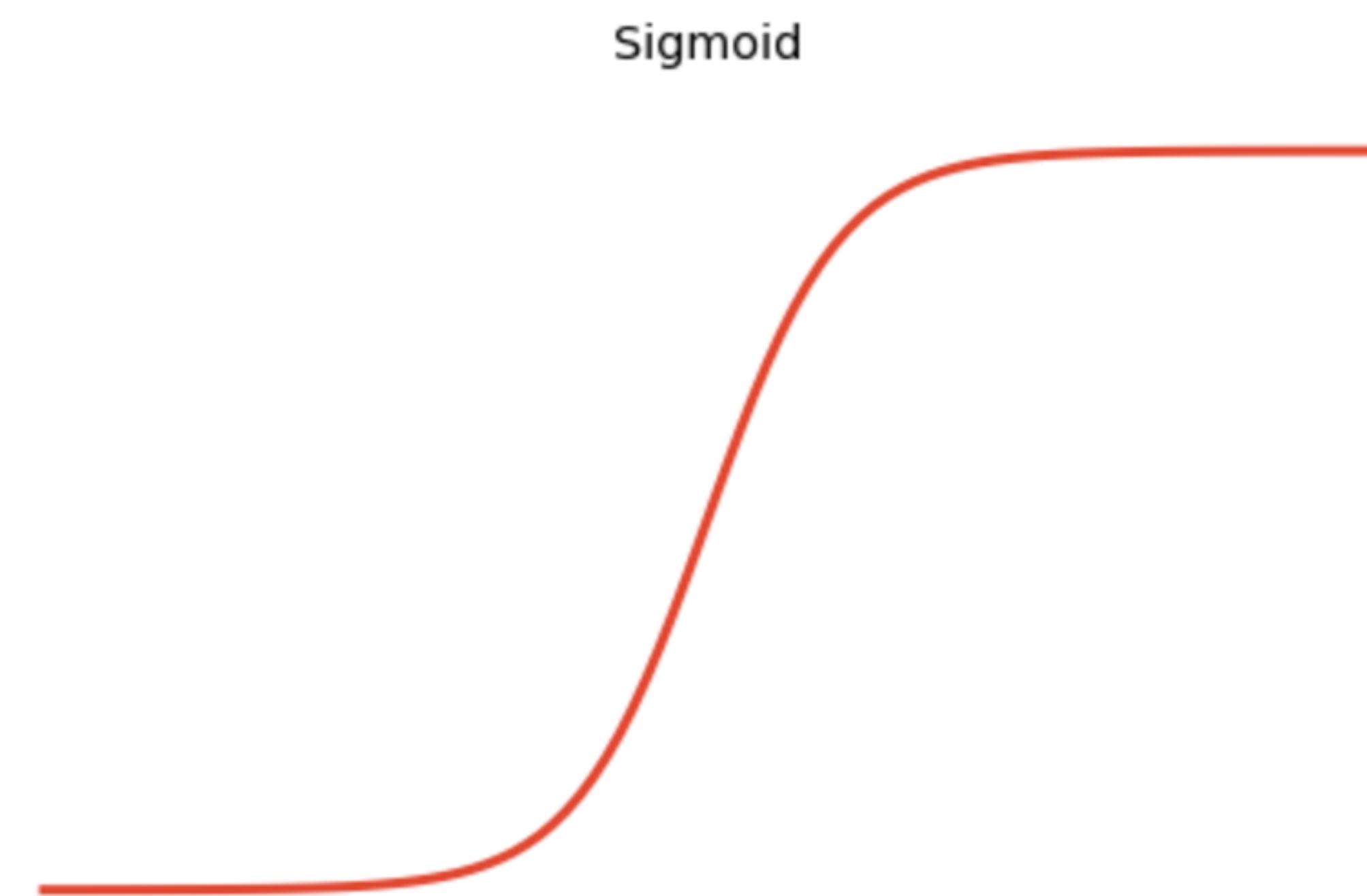


```
In [12]: plot_activation(lambda x: 0.5 + 0.1*x, title='Linear') # example
```



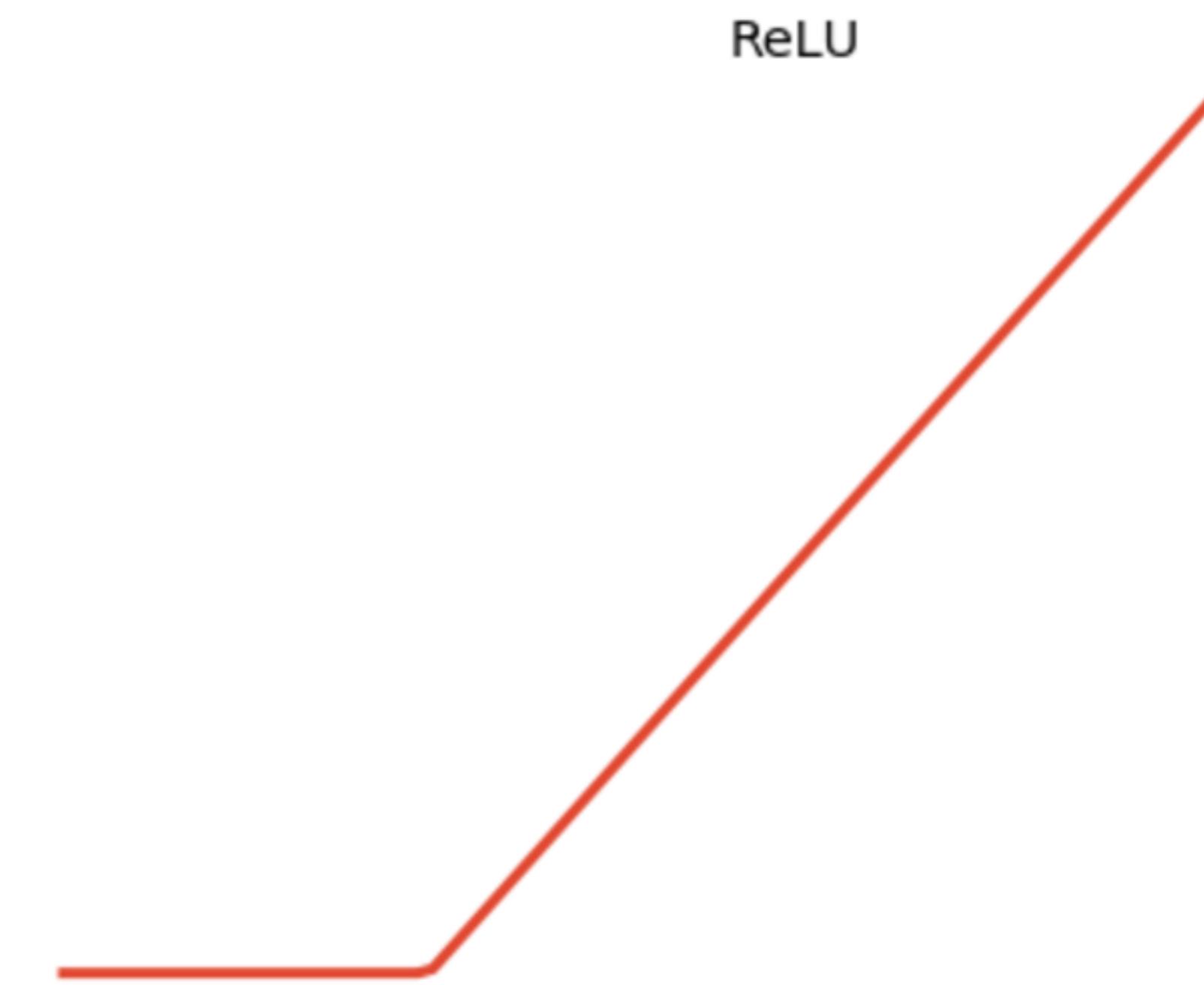
```
In [13]: def sigmoid_layer(input_tensor, num_units):
    """Sigmoid activation layer: output = sigmoid(inputs * weights + biases)"""
    return tf.nn.sigmoid(linear_layer(input_tensor, num_units), name='sigmoid')

plot_activation(lambda x: 1.0 / (1.0 + np.exp(-x))), title='Sigmoid' # example
```



```
In [14]: def relu_layer(input_tensor, num_units):
    """ReLU activation layer: output = ReLU(inputs * weights + biases)"""
    return tf.nn.relu(linear_layer(input_tensor, num_units), name='relu')

plot_activation(lambda x: (0.5 + 0.1*x).clip(min=0), title='ReLU') # example
```



Create network structure

```
In [15]: # Let's make a network with 2 hidden layers (and an output layer)
hidden1_num_units = 10
hidden2_num_units = 3
output_num_units = 1 # binary classification needs only 1 output

with tf.name_scope('hidden1'):
    hidden1 = relu_layer(X_placeholder, hidden1_num_units)

with tf.name_scope('hidden2'):
    hidden2 = relu_layer(hidden1, hidden2_num_units)

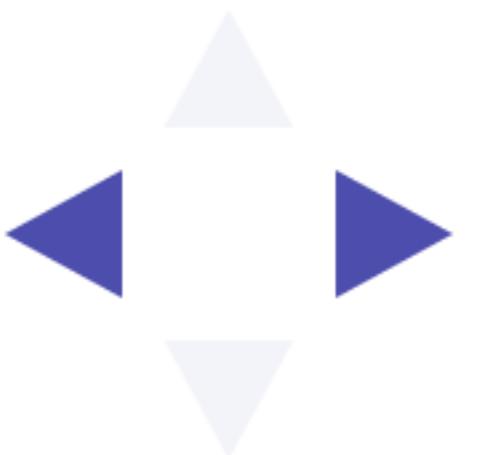
with tf.name_scope('output'):
    output = sigmoid_layer(hidden2, output_num_units)
```



Choose an error metric

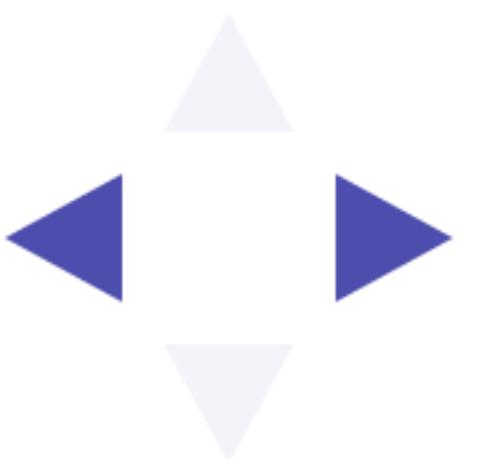
```
In [16]: def l2_loss(logits, labels):
    """Euclidean distance or L2-norm: sqrt(sum((logits - labels)^2))"""
    labels = tf.to_float(labels)
    return tf.nn.l2_loss(logits - labels, name='l2_loss')

with tf.name_scope('error'):
    error = l2_loss(output, y_placeholder) # predicted vs. true labels
    tf.summary.scalar(error.op.name, error) # write error (loss) to log
```



```
In [18]: print("Nodes in computation graph:")
+ "\n  Input features: {}".format(X_placeholder)
+ "\n  Hidden layer 1: {}".format(hidden1)
+ "\n  Hidden layer 2: {}".format(hidden2)
+ "\n  Output labels : {}".format(output)
+ "\n  Ground truth   : {}".format(y_placeholder)
+ "\n  Error metric   : {}".format(error))
```

```
Nodes in computation graph:
Input features: Tensor("X:0", shape=(?, 2), dtype=float32)
Hidden layer 1: Tensor("hidden1/relu:0", shape=(?, 10), dtype=float32)
Hidden layer 2: Tensor("hidden2/relu:0", shape=(?, 3), dtype=float32)
Output labels : Tensor("output/sigmoid:0", shape=(?, 1), dtype=float32)
Ground truth   : Tensor("y:0", shape=(?, 1), dtype=int64)
Error metric   : Tensor("error/l2_loss:0", shape=(), dtype=float32)
```



Setup Logging

These logs are later read and visualized by TensorBoard.

```
In [19]: import time
import os

log_basedir = "logs"
run_label = time.strftime('%Y-%m-%d_%H-%M-%S') # e.g. 2016-08-18_21-30-45
log_path = os.path.join(log_basedir, run_label)
all_summaries = tf.summary.merge_all()
summary_writer = tf.summary.FileWriter(log_path, session.graph)
print("Logging to: {}".format(log_path))
```

Logging to: logs/2017-05-04_08-18-39



Train your Model

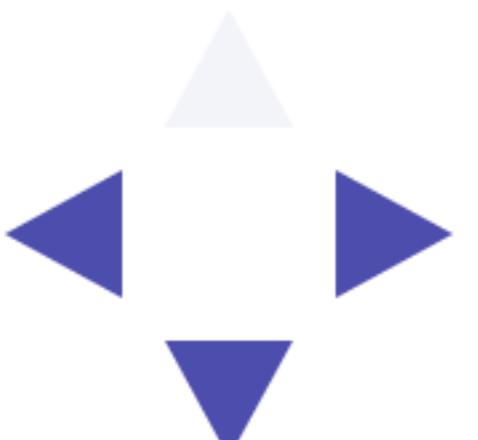
In [20]:

```
# Pick a training algorithm
def sgd_train(error, learning_rate=0.01):
    """Gradient descent optimizer for training.

    Creates an optimizer to compute and apply gradients to all trainable variables.

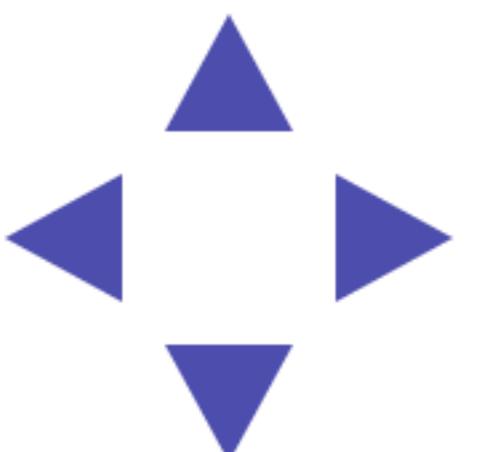
    Args:
        error: Error (loss) metric.
        learning_rate: Controls the size of each step the optimizer takes.
    Returns:
        training: Training operation, ready to be called with tf.Session.run().
    """
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    return optimizer.minimize(error)

with tf.name_scope('training'):
    training = sgd_train(error)
```



```
In [21]: # Define training parameters
num_steps = 1000 # how many iterations to train for
batch_size = 100 # how many samples in each iteration

# Initialize variables
init_op = tf.global_variables_initializer()
session.run(init_op)
```



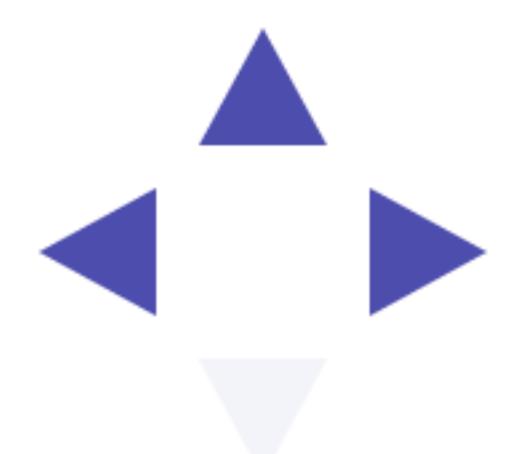
In [22]:

```
# Run training operation for num_steps iterations
for step in range(num_steps):
    # Randomly pick batch_size samples from training set
    sample_idx = np.random.choice(len(X_train), batch_size, replace=False)
    feed_dict = {
        X_placeholder: X_train[sample_idx, :],
        y_placeholder: y_train[sample_idx, :]
    }
    # Note: feed_dict uses placeholder objects as key!

    # Train for one iteration, time it
    start_time = time.time()
    _, error_value = session.run([training, error], feed_dict=feed_dict)
    duration = time.time() - start_time

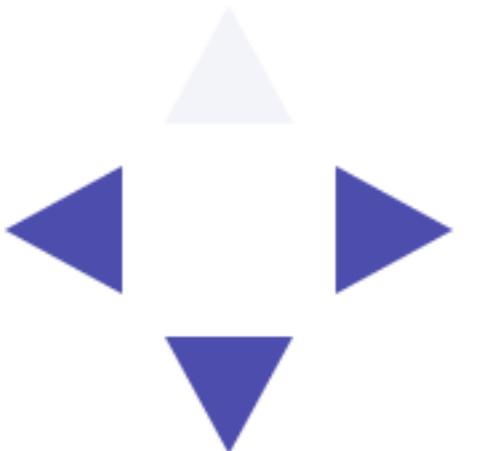
    # Print an overview and write summaries (logs) every 100 iterations
    if step % 100 == 0 or step == (num_steps - 1):
        print("Step {:4d}: training error = {:.5f} {:.3f} sec"
              .format(step, error_value, duration))
        summary_str = session.run(all_summaries, feed_dict=feed_dict)
        summary_writer.add_summary(summary_str, step)
        summary_writer.flush()
```

```
Step 0: training error = 22.94 (0.007 sec)
Step 100: training error = 2.20 (0.001 sec)
Step 200: training error = 1.62 (0.001 sec)
Step 300: training error = 1.74 (0.001 sec)
Step 400: training error = 0.88 (0.002 sec)
Step 500: training error = 2.45 (0.001 sec)
Step 600: training error = 1.34 (0.001 sec)
Step 700: training error = 1.47 (0.001 sec)
Step 800: training error = 0.77 (0.001 sec)
Step 900: training error = 1.45 (0.001 sec)
Step 999: training error = 1.08 (0.001 sec)
```



Test your Model

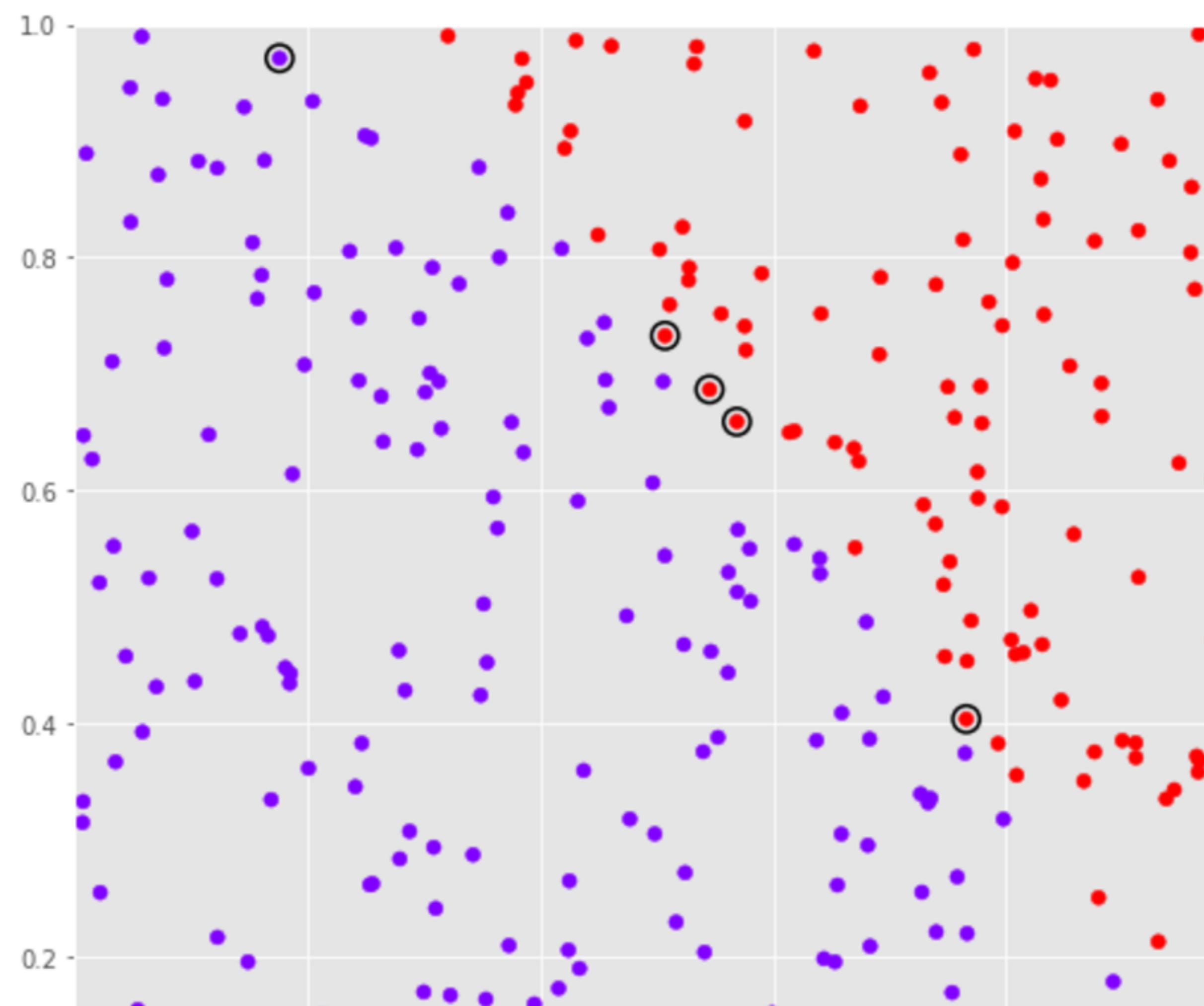
```
In [23]: # Check performance on test set
y_test_pred, test_error = session.run([output, error], feed_dict={
    x_placeholder: x_test,
    y_placeholder: y_test
})
# Note: The placeholder shapes must be compatible with the tensors being supplied!
```



```
In [24]: y_test_pred = np.int_(np.round_(y_test_pred))
mismatches = (y_test_pred - y_test).flat != 0
print("Test error = {:.2f} ({}) mismatches"
      .format(test_error, sum(np.int_(mismatches))))
_, ax = plot_data_2D(X_test, c=y_test_pred)
ax.scatter(X_test[mismatches, 0], X_test[mismatches, 1],
           s=128, marker='o', facecolors='none', edgecolors='black', linewidths=1.5)
```

Test error = 2.93 (6 mismatches)

Out[24]: <matplotlib.collections.PathCollection at 0x10f209ba8>

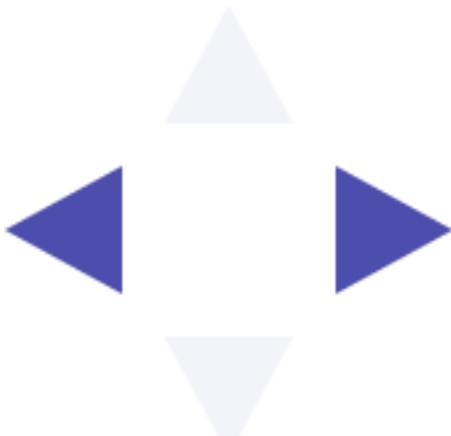


Visualize using TensorBoard

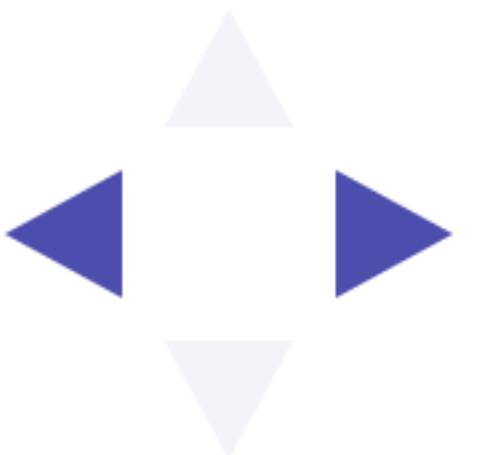
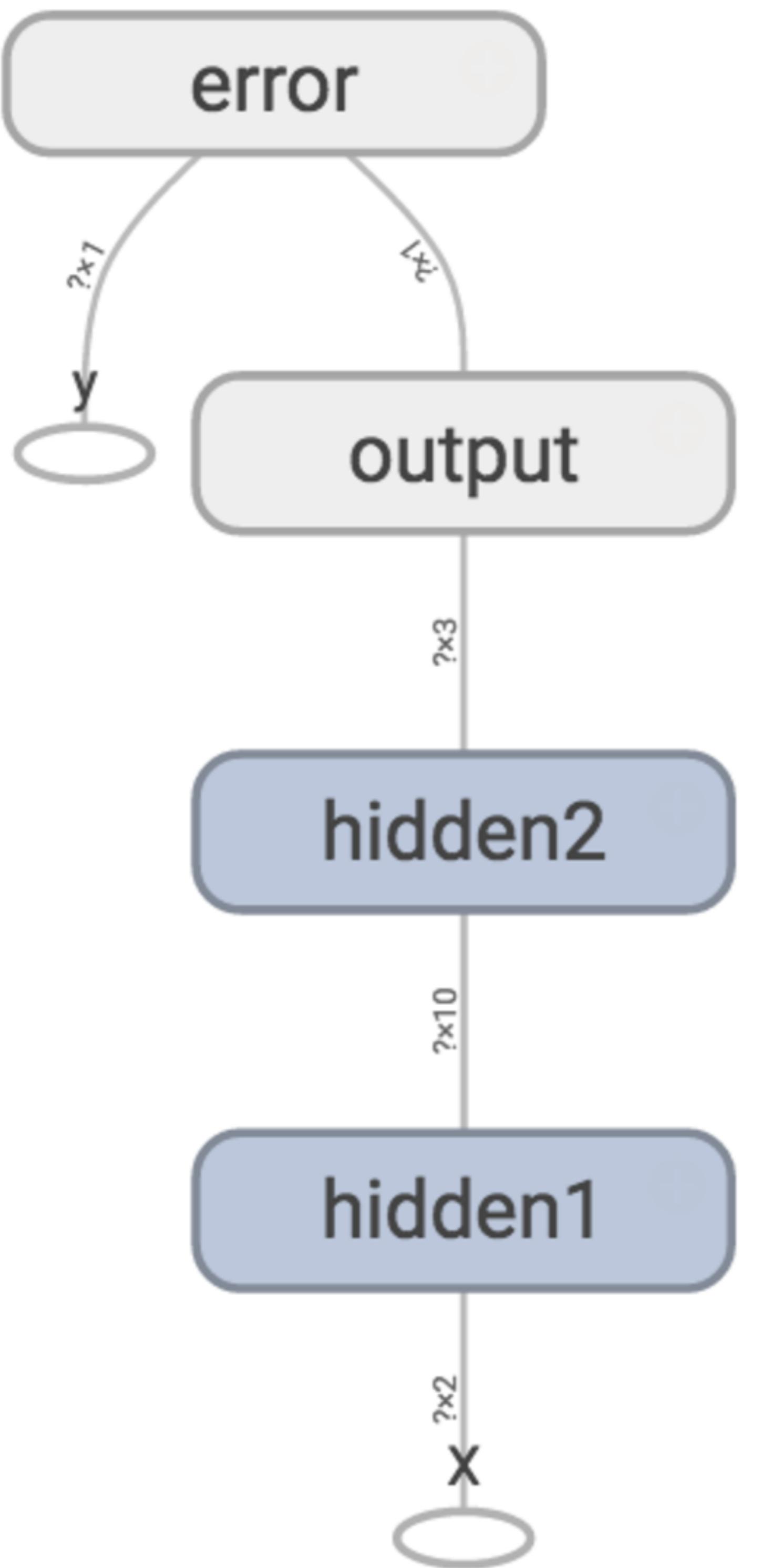
Run TensorBoard with the following command, passing in the appropriate log directory:

```
tensorboard --logdir=logs
```

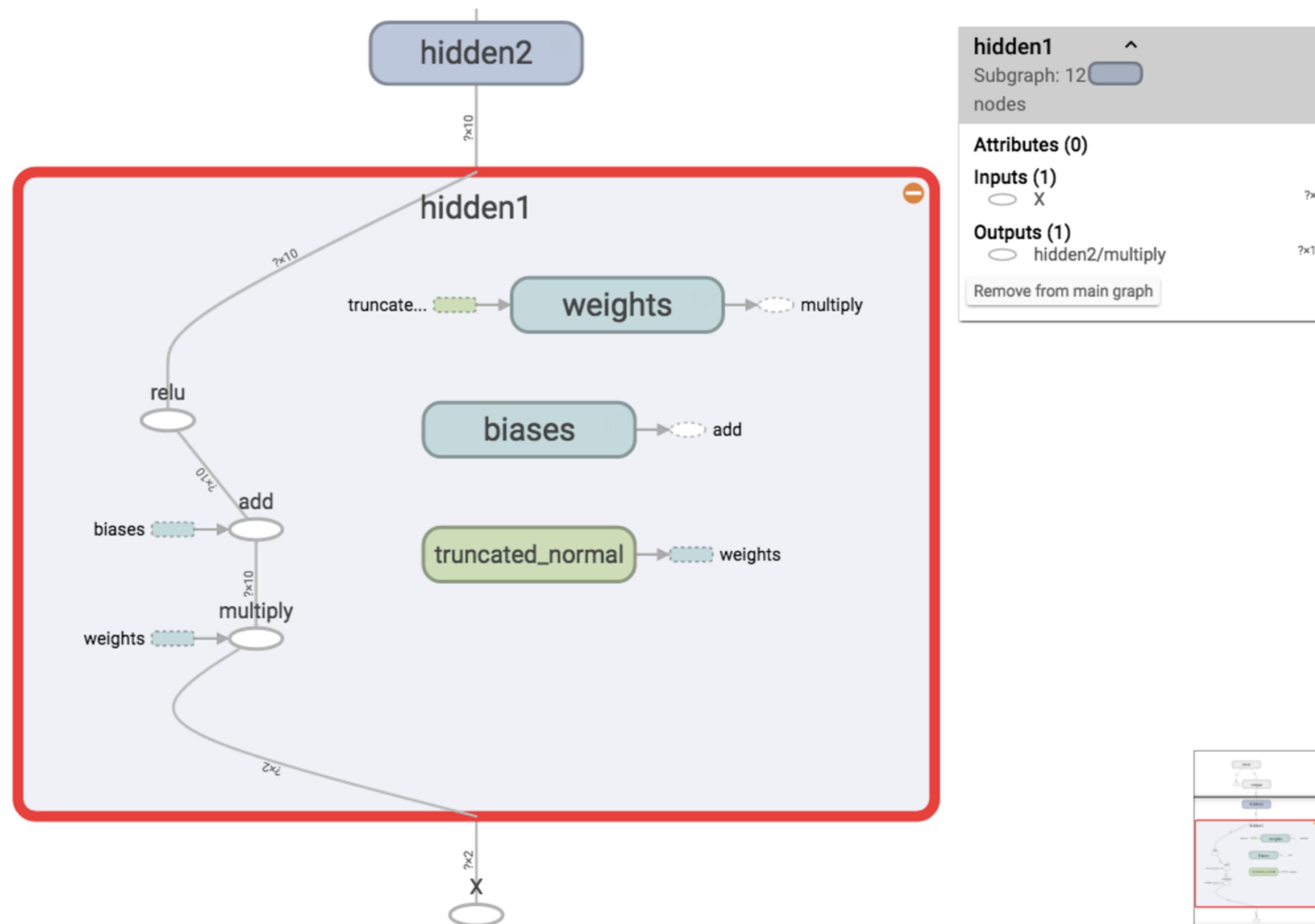
And then open the URL that gets printed, in your browser (typically: <http://0.0.0.0:6006>).



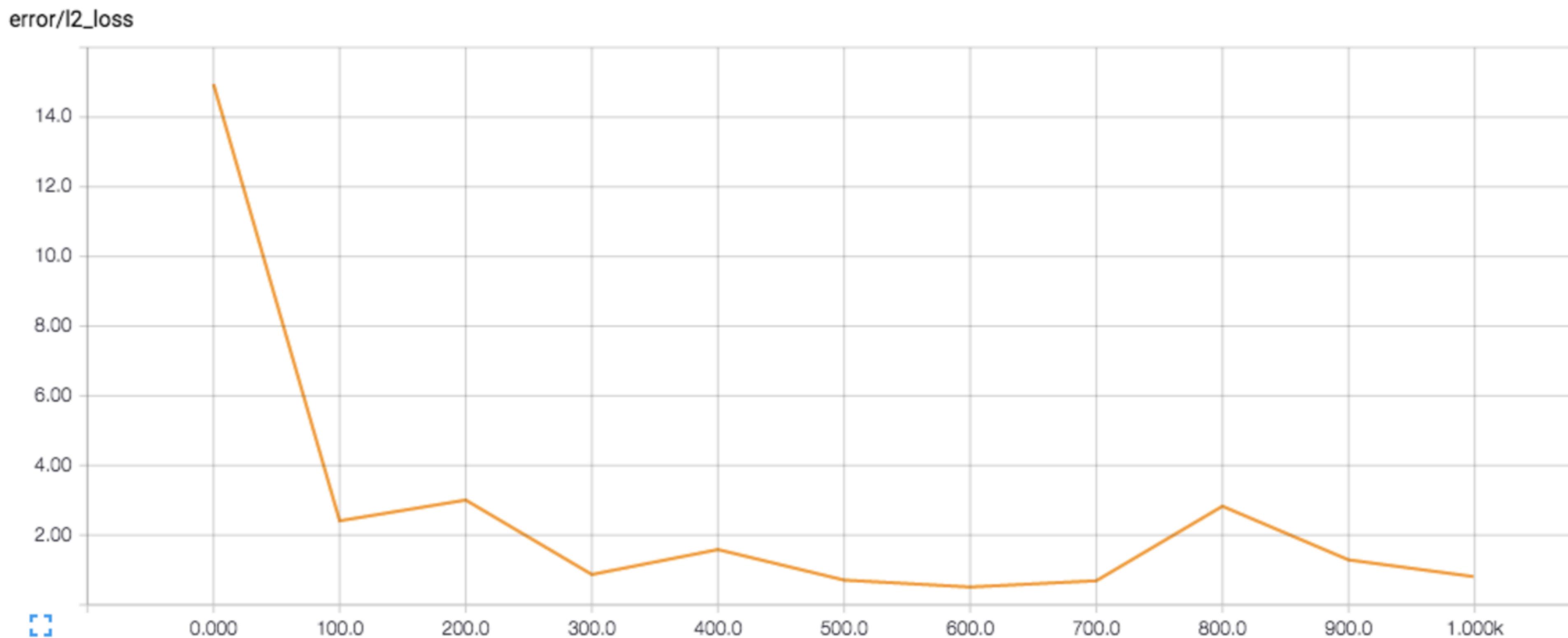
Computation graph



Layer detail: hidden1



Error plot

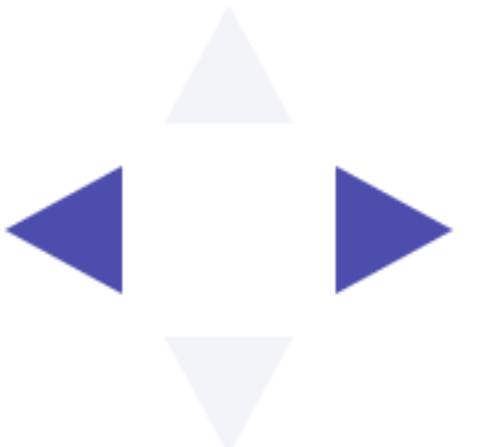


Summary

- Define a neural network model as nodes of a computation graph in TensorFlow.
- Train your model by presenting labeled data and applying an optimization algorithm.
- Log important values like error during training.
- Test your resulting model on unseen data.
- Visualize computation graph and logged data using TensorBoard.

Note: You can convert this notebook into slides (HTML + reveal.js) and serve them using:

```
jupyter nbconvert tensorboard_basics.ipynb --to slides --post serve
```



Deep Neural Networks using TensorBoard

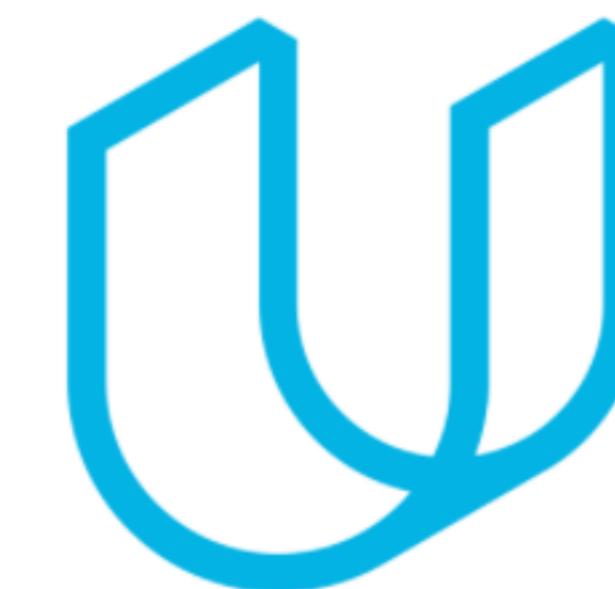
github.com/PythonWorkshop/tensorboard_demos

Luis Serrano

[@luis likes math](https://@luis_likes_math)
github.com/luisguiserrano

Arpan Chakraborty

@runoffthemill
github.com/napratin



U D A C I T Y

udacity.com/ai

udacity.com/ml

