TensorFlow Tutorial #01

Linear Regression

Seonghyeon Nam, Ph.D. Student
Computational Intelligence and Photography Lab.
Yonsei University

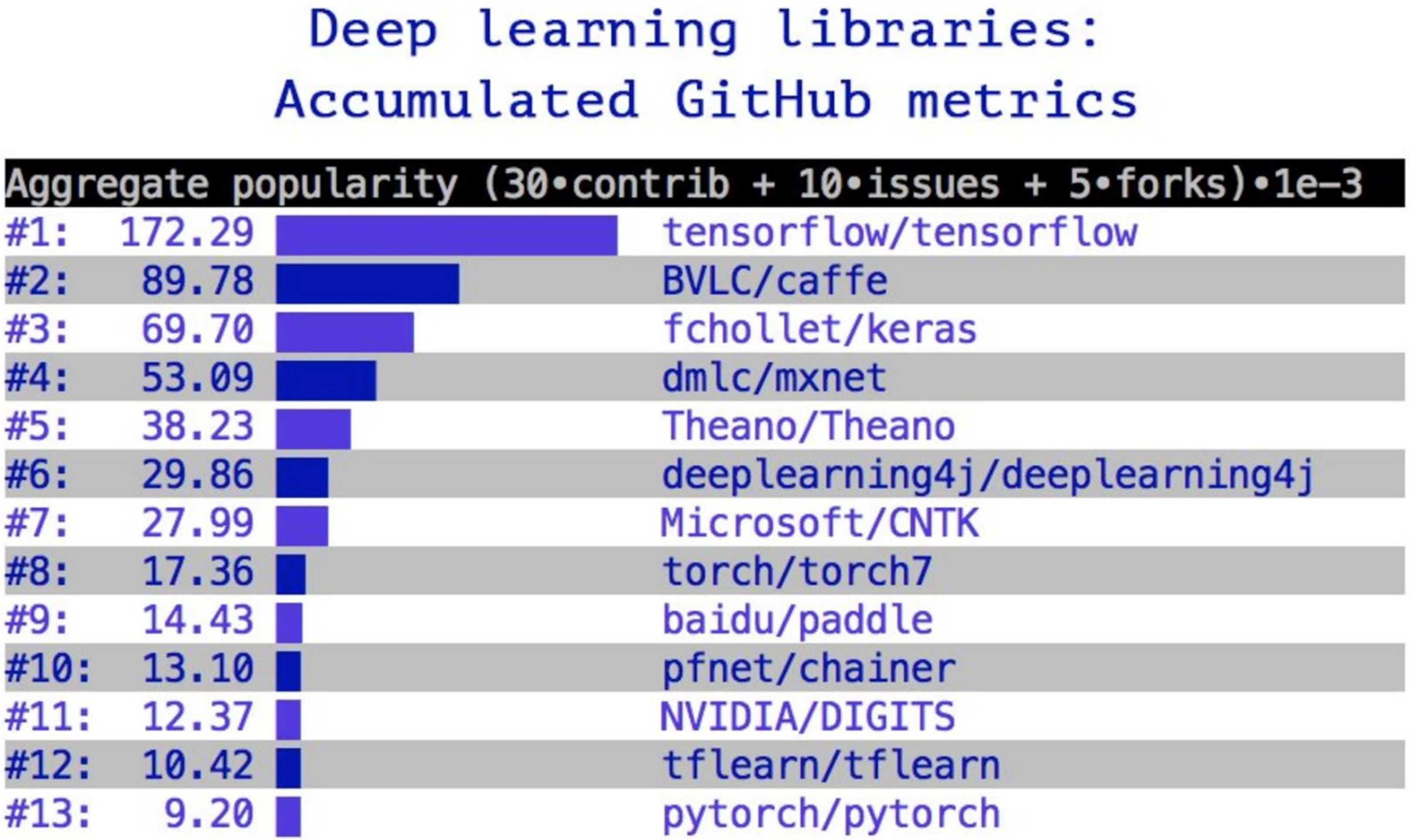
Acknowledgement

1. TensorFlow website
   http://www.tensorflow.org
2. CS20SI: TensorFlow for Deep Learning Research
   http://web.stanford.edu/class/cs20si/
3. Hun Kim, DeepLearningZeroToAll
   http://hunkim.github.io/ml/

TensorFlow?

- Open source library for numerical computation using data flow graphs
- Developed by Google Brain Team
- Provides various functions and classes to implement machine learning and deep neural networks

Why TensorFlow?



https://twitter.com/fchollet/status/830499993450450944/

Tensors

The central unit of data in TensorFlow is the **tensor**.
NOTE: Rank == the number of dimsensions

```
# a rank 0 tensor; this is a scalar with shape []
3

# a rank 1 tensor; this is a vector with shape [3]
[1. ,2., 3.]

# a rank 2 tensor; a matrix with shape [2, 3]
[[1., 2., 3.], [4., 5., 6.]]

# a rank 3 tensor with shape [2, 1, 3]
[[[1., 2., 3.]], [[7., 8., 9.]]]
```

Tensors

Rank	Math entity	Python example
0	Scalar (magnitude only)	`s = 483`
1	Vector (magnitude and direction)	`v = [1.1, 2.2, 3.3]`
2	Matrix (table of numbers)	`m = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`
3	3-Tensor (cube of numbers)	`t = [[[2], [4], [6]], [[8], [10], [12]], [[14], [16], [18]]]`
n	n-Tensor (you get the idea)	`....`

Tensors

Rank	Shape	Dimension number	Example
0	[]	0-D	A 0-D tensor. A scalar.
1	[D0]	1-D	A 1-D tensor with shape [5].
2	[D0, D1]	2-D	A 2-D tensor with shape [3, 4].
3	[D0, D1, D2]	3-D	A 3-D tensor with shape [1, 4, 3].
n	[D0, D1, ... Dn-1]	n-D	A tensor with shape [D0, D1, ... Dn-1].

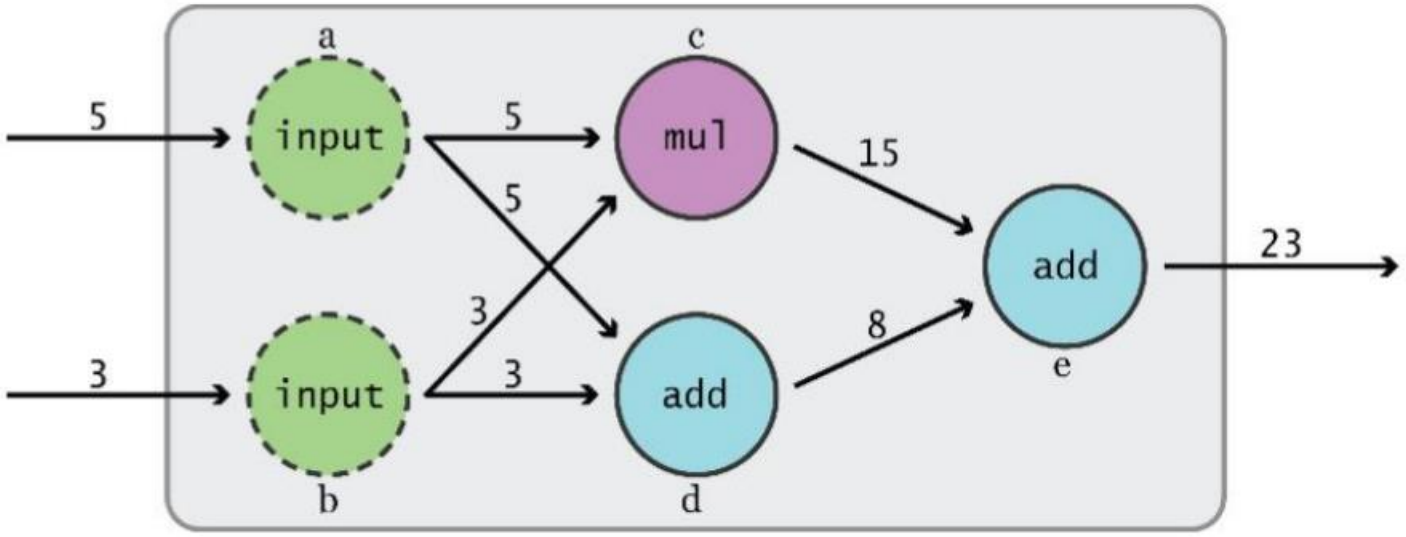
Tensors

Data type	Python type	Description
`DT_FLOAT`	`tf.float32`	32 bits floating point.
`DT_DOUBLE`	`tf.float64`	64 bits floating point.
`DT_INT8`	`tf.int8`	8 bits signed integer.
`DT_INT16`	`tf.int16`	16 bits signed integer.
`DT_INT32`	`tf.int32`	32 bits signed integer.
`DT_INT64`	`tf.int64`	64 bits signed integer.

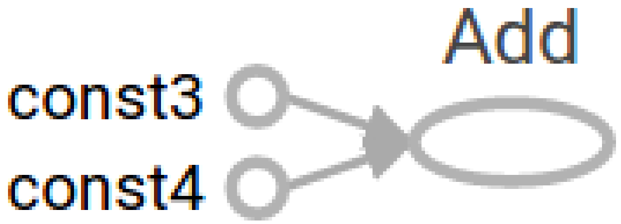
Computational Graph

… also known as Data Flow Graph

TensorFlow separates definition of computations from their execution



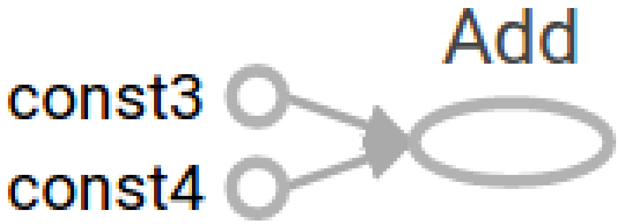
Phase 1: Definition of a Computational Graph




```
import tensorflow as tf

node1 = tf.constant(3.0, tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly
print(node1, node2)

>> Tensor("Const:0", shape=(), dtype=float32) Tensor("Const_1:0", shape=(), dtype=float32)

node3 = tf.add(node1, node2)
print("node3: ", node3) # are you expecting 7?

>> node3: Tensor("Add_2:0", shape=(), dtype=float32) # actually it’s not 7 :(
```

The output of Print() is the computational nodes, not the numerical values 3, 4, 7.

Phase 2: Execution Using a Session

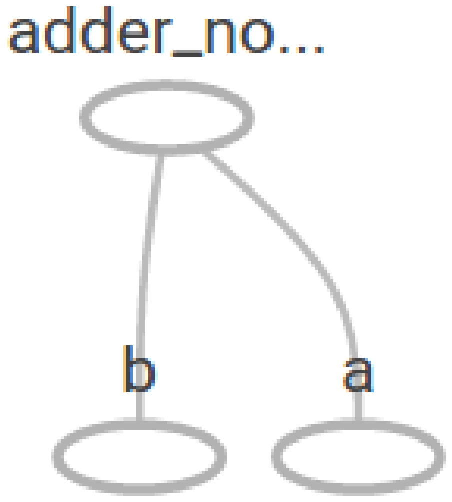



```
import tensorflow as tf

node1 = tf.constant(3.0, tf.float32)
node2 = tf.constant(4.0) # also tf.float32 implicitly

sess = tf.Session()
print(sess.run([node1, node2]))

>> [3.0, 4.0]

node3 = tf.add(node1, node2)
print(“sess.run(node3): ", sess.run(node3))

>> sess.run(node3): 7.0
```

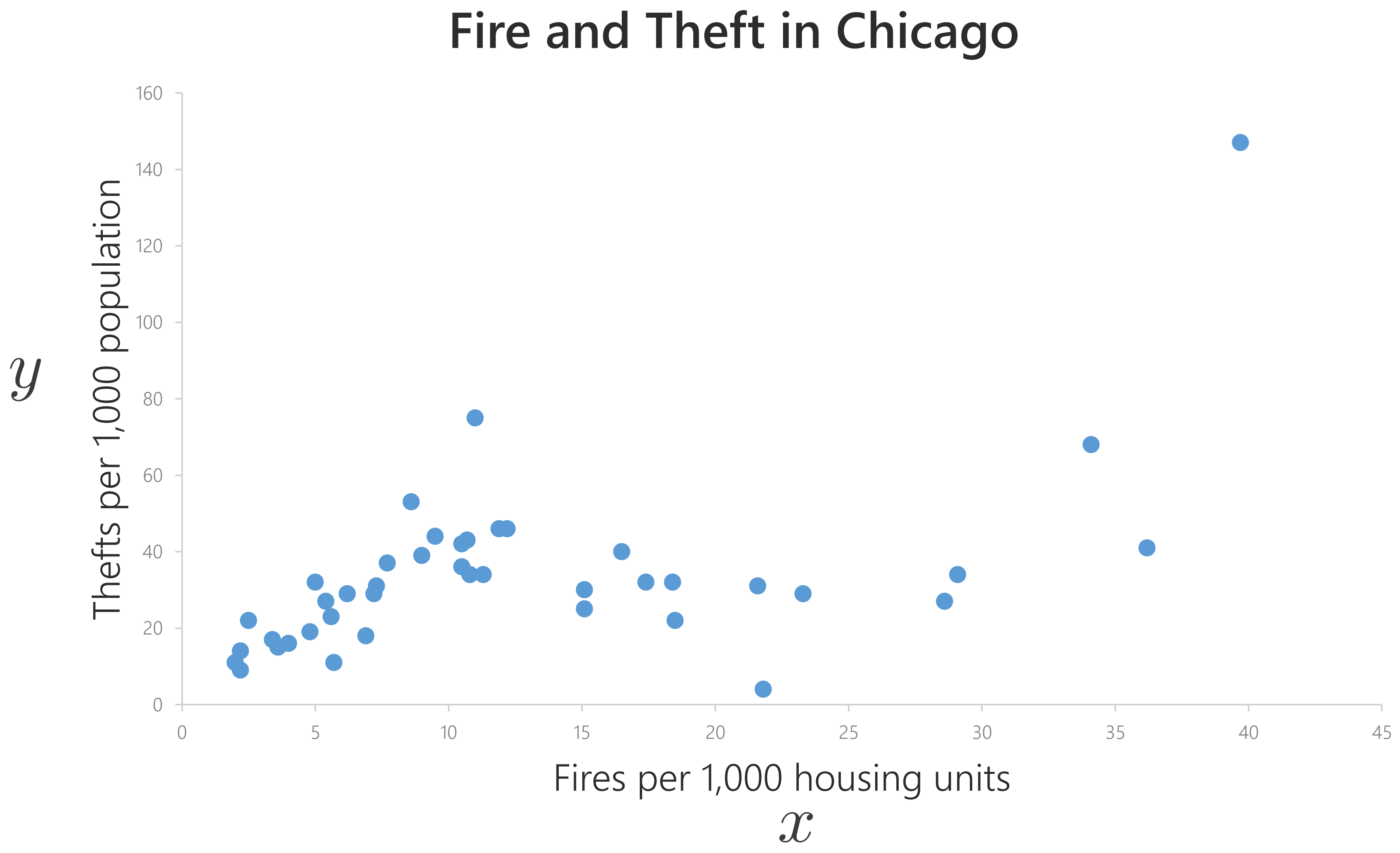
Placeholder

We want to feed our data into the computational graph

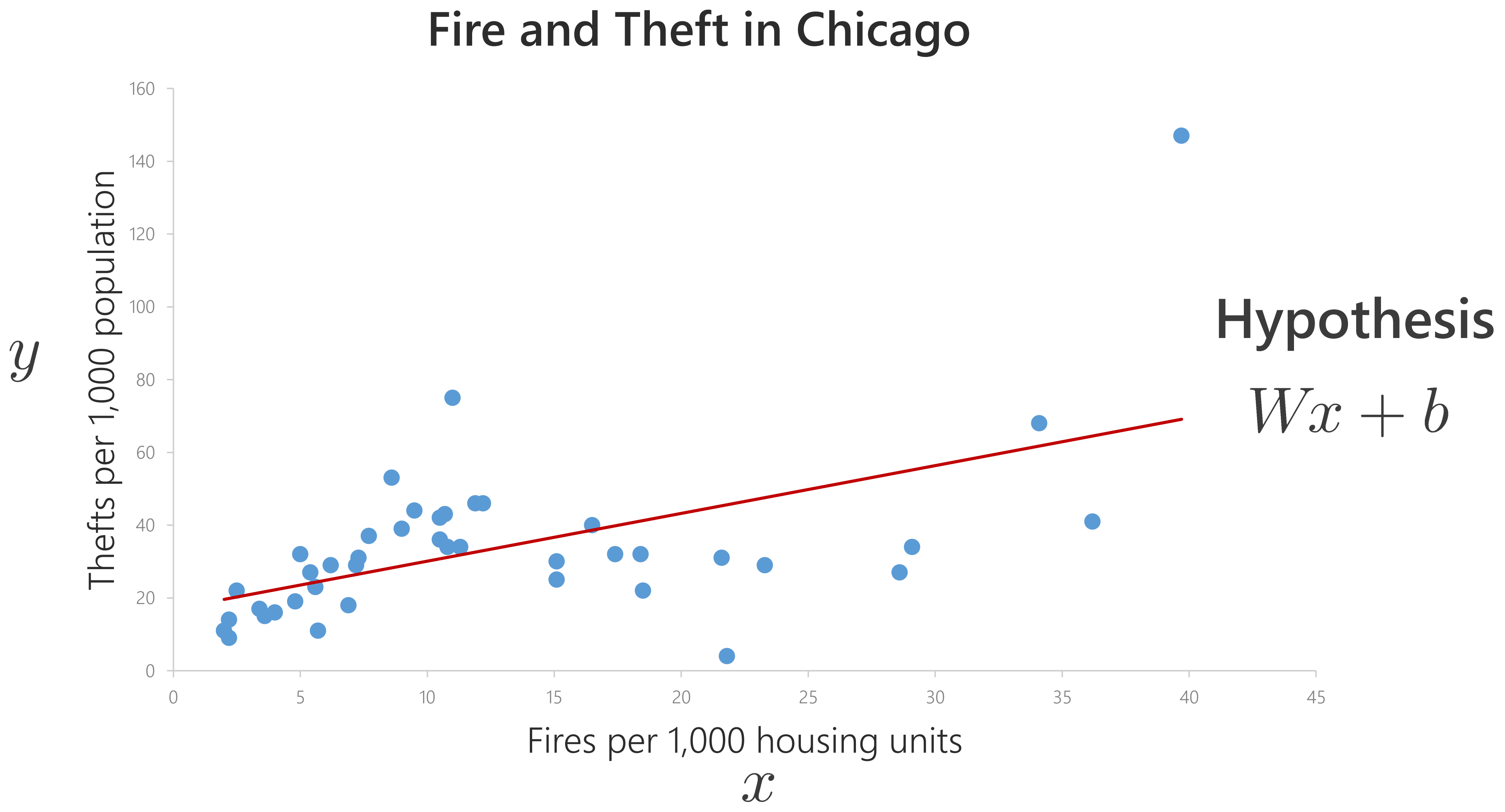
```
import tensorflow as tf

a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)
adder_node = a + b # + provides a shortcut for tf.add(a, b)

sess = tf.Session()
print(sess.run(adder_node, {a: 3, b: 4.5}))
print(sess.run(adder_node, {a: [1, 3], b: [2, 4]}))

>> 7.5
   [3. 7.]
```



Linear Regression



Linear Regression



Optimization

Our hypothesis for modeling data is

$$H(x) = Wx + b$$

To find the optimal **W** and **b**, we minimize the following cost function

$$cost\left(W, b\right) = \frac{1}{N}\sum_{i=1}^{N}(H(x^i) - y^i)^2$$

Building a TF Graph

```
# Step 1: Load data
data = ...

# Step 2: create placeholders for input X (number of fires) and label Y (number of thefts)
X = tf.placeholder(tf.float32, name='X')
Y = tf.placeholder(tf.float32, name='Y')

# Step 3: create weight and bias, initialized to 0
w = tf.Variable(0.0, name='weights')
b = tf.Variable(0.0, name='bias')

# Step 4: build model to predict Y
Y_predicted = X * w + b
```

Trainable variables

$$H(x) = Wx + b$$

Building a TF Graph

```
# Step 5: use the square error as the loss function
loss = tf.reduce_mean(tf.square(Y - Y_predicted, name='loss'))
```

$$cost\left(W,b\right) = \frac{1}{N}\sum_{i=1}^{N}(H(x^i) - y^i)^2$$

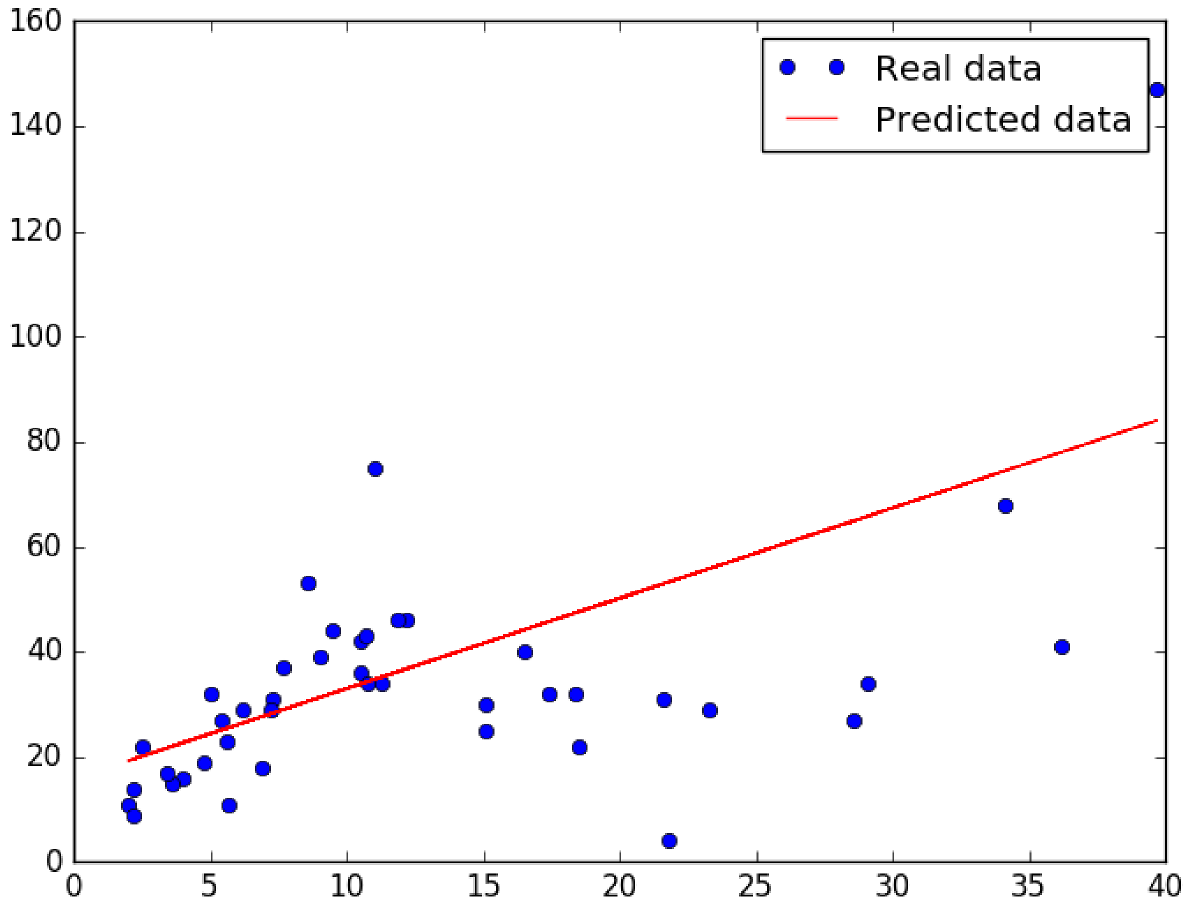
```
# Step 6: using gradient descent with learning rate of 0.01 to minimize loss
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
train = optimizer.minimize(loss)
```

Training

```python
with tf.Session() as sess:
    # Step 7: initialize w and b
    sess.run(tf.global_variables_initializer())

    # Step 8: train the model
    for i in range(100):  # train the model 100 times
        total_loss = 0
        for x, y in data:
            # Session runs train_op and fetch values of loss
            _, l = sess.run([train, loss], feed_dict={X: x, Y: y})
            total_loss += l
        print('Epoch {0}: {1}'.format(i, total_loss / n_samples))

    # Step 9: output the values of w and b
    w_value, b_value = sess.run([w, b])
```

Training

```
>> ...
...
...
Epoch 89: 1426.038033108981
Epoch 90: 1424.5748210840281
Epoch 91: 1423.1531702368743
Epoch 92: 1421.771026852585
Epoch 93: 1420.4274983895677
Epoch 94: 1419.121967994741
Epoch 95: 1417.85251878131
Epoch 96: 1416.618930517208
Epoch 97: 1415.4196022436731
Epoch 98: 1414.2534379121803
Epoch 99: 1413.1202843011845
```



Multivariate Linear Regression

x^1	x^2	x^3	**Y**
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

$$H\left(x_1, x_2, x_3\right)$$

$$= w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

$$= \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix}^T + b$$

$$= WX + b$$

Building a TF Graph

```
# Step 1: Load data
data = ...

# Step 2: create placeholders
X = tf.placeholder(tf.float32, shape=[None, 3], name='X')
Y = tf.placeholder(tf.float32, shape=[None, 1], name='Y')

# Step 3: create weight and bias
W = tf.Variable(tf.random_normal([3, 1]), name='weights')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Step 4: build model to predict Y
Y_predicted = tf.matmul(X, W) + b
```

Training is similar to that of the previous case...