Introduction to TensorFlow

Hackathon 2018

Part 1 – The basics

What is TensorFlow?

Software library for numerical computation

… but wait, why not just use Numpy?

TensorFlow vs Numpy

Easier to run on GPUs

TensorFlow vs Numpy

Numpy:
Expensive computations are done outside of python

Problem:
Overhead for switching in and out of python for every operation.

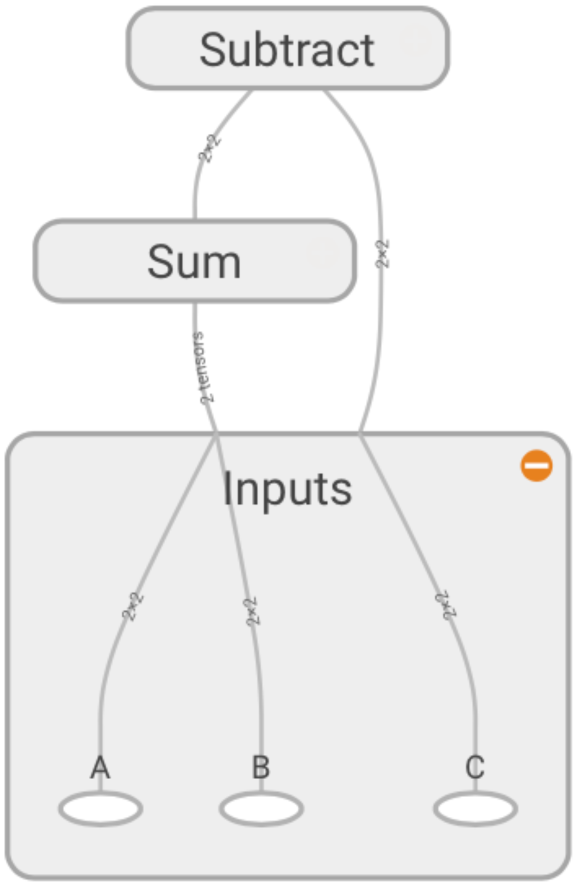
TensorFlow:
Sets of interacting operations that can be run all outside of python.

Data Flow Graphs

Representations of the data dependencies between a number of operations

```
Sum = A + B

Subtract = Sum - C
```



The Basics

There are 2 main parts to a TensorFlow program:

- Building the graph
- Running the graph

Building the Graph

Give a name
to the graph

```
import tensorflow as tf

graph = tf.Graph()
with graph.as_default():
    a = tf.placeholder(dtype=tf.int32, shape=[1])
    b = tf.placeholder(dtype=tf.int32, shape=[1])
    sum_ab = tf.add(a, b)
```

Building the Graph

Say what shape and type your data will be

```
import tensorflow as tf

graph = tf.Graph()
with graph.as_default():
    a = tf.placeholder(dtype=tf.int32, shape=[1])
    b = tf.placeholder(dtype=tf.int32, shape=[1])
    sum_ab = tf.add(a, b)
```

Building the Graph

```
import tensorflow as tf

graph = tf.Graph()
with graph.as_default():
    a = tf.placeholder(dtype=tf.int32, shape=[1])
    b = tf.placeholder(dtype=tf.int32, shape=[1])
    sum_ab = tf.add(a, b)
```

Define the operations that you want to have

Running the Graph

Define a session

```
with tf.Session(graph=graph) as sess:
    result = sess.run(sum_ab, feed_dict={a:[1], b:[2]})
```

Running the Graph

```
with tf.Session(graph=graph) as sess:
    result = sess.run(sum_ab, feed_dict={a:[1], b:[2]})
```

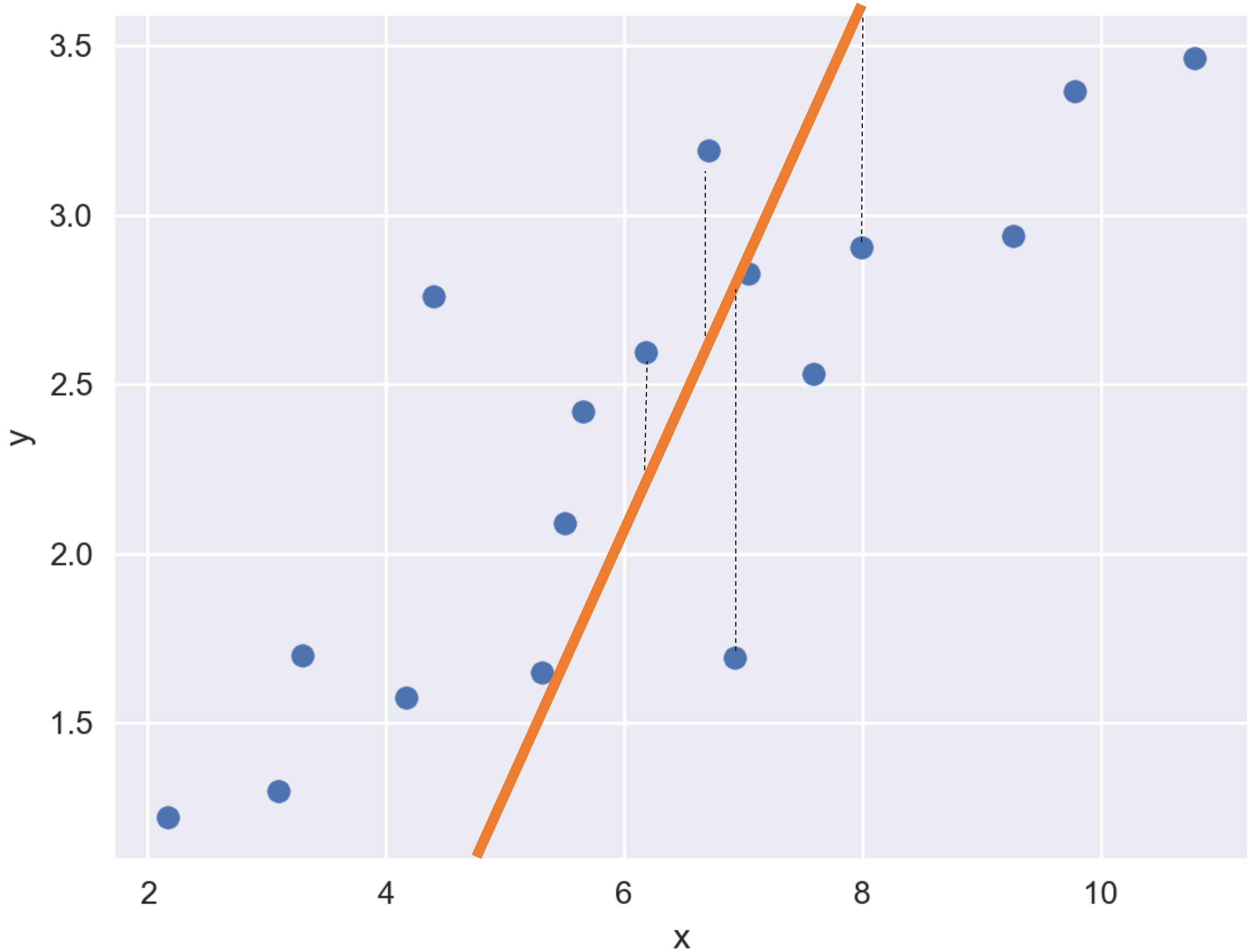
Run the operations

Introduction to TensorFlow

Hackathon 2018

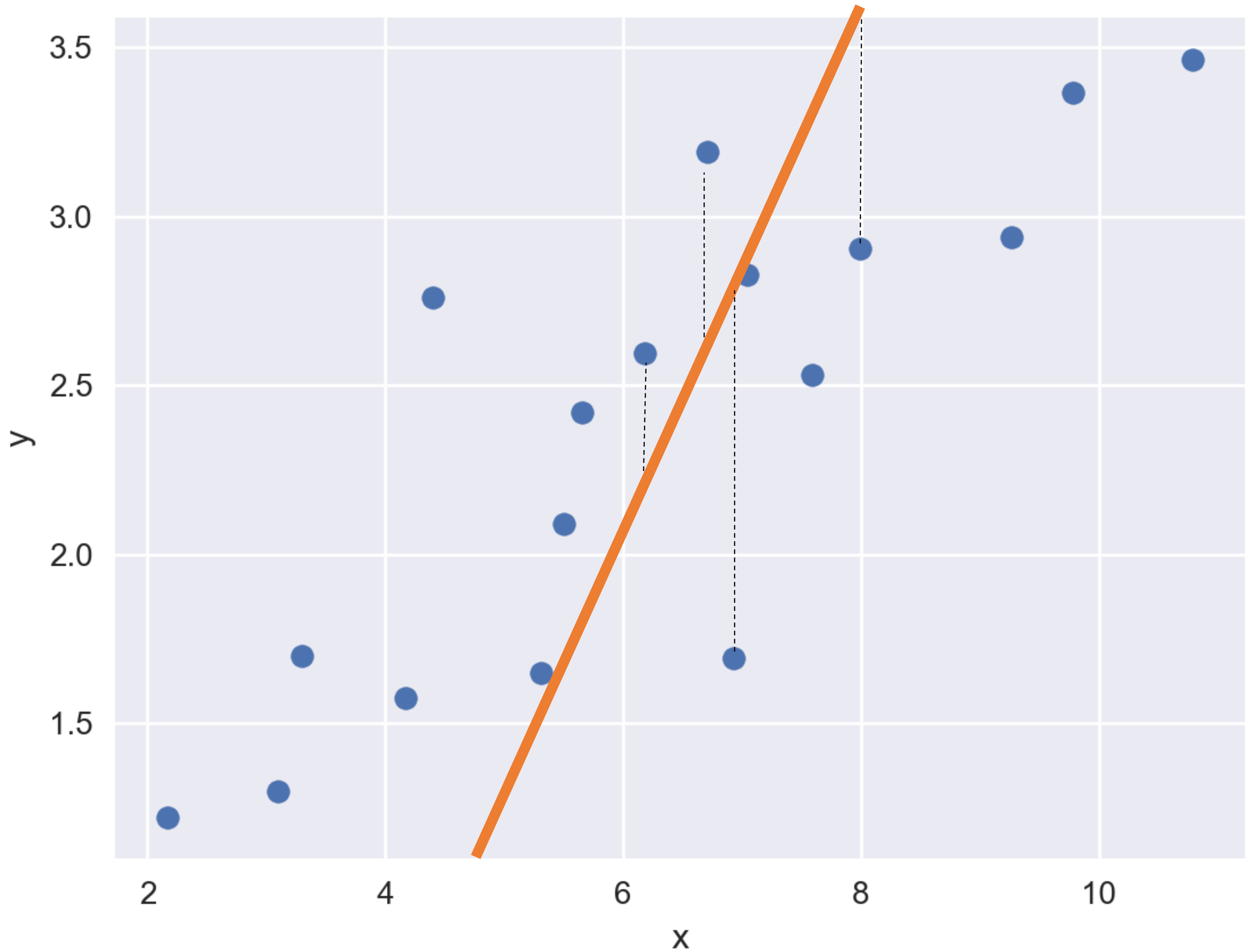
Part 2 – Building on the basics

Linear Regression



$$y = mx + c$$

Linear Regression

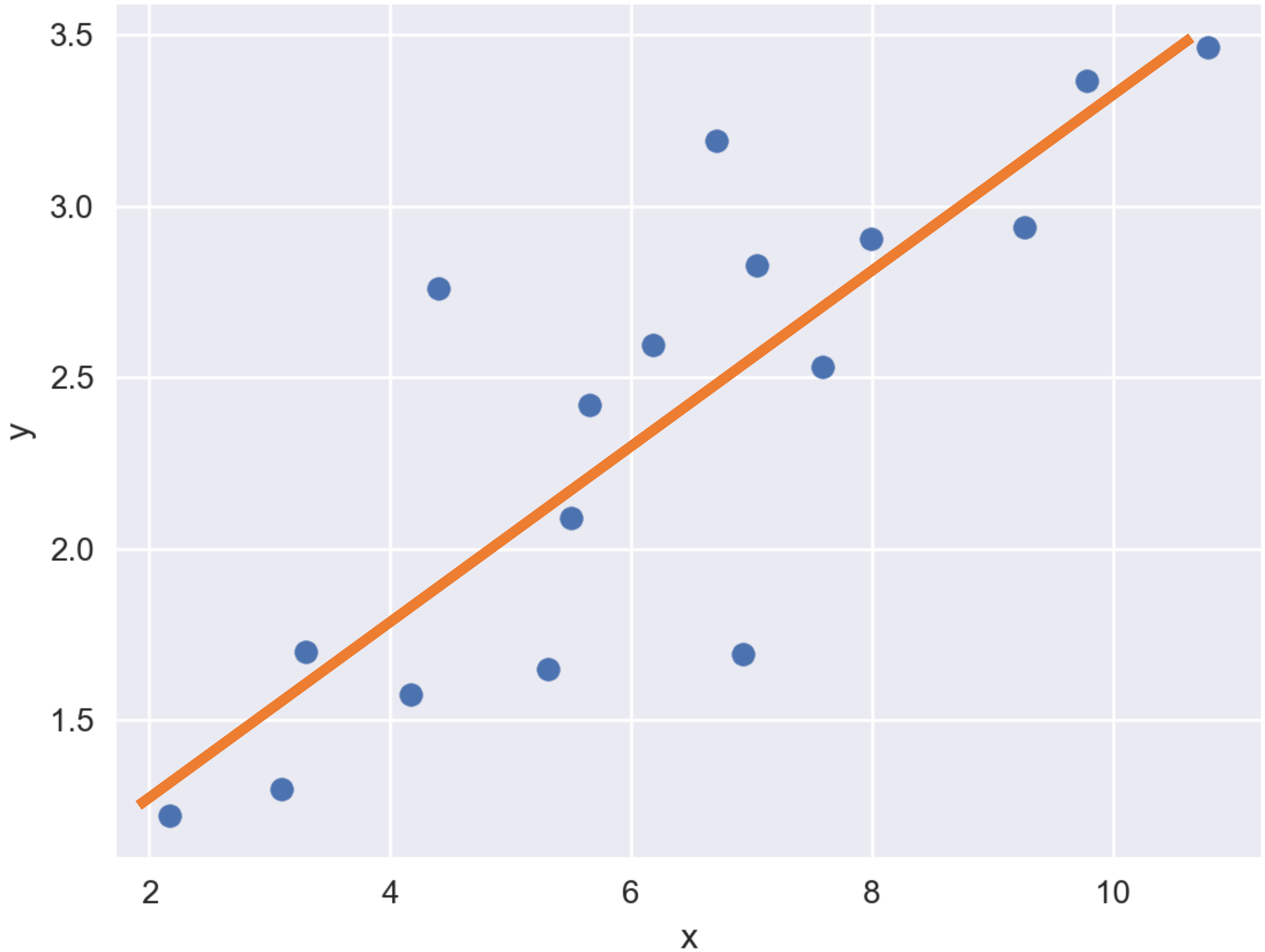


Cost function:

$$cost = \frac{1}{N}\sum_{i=1}^{N}\left(Y_N^{Model} - Y_N^{Data}\right)^2$$

Number of data points

Linear Regression



$$y = mx + c$$

Linear Regression

```
# Placeholders - where the data can come into the graph
X = tf.placeholder(tf.float32, [None])
Y = tf.placeholder(tf.float32, [None])
```

Can be a variable size

Linear regression

```
# Creating the parameters - theta1 is the slope, theta0 is the intercept
theta0 = tf.Variable(np.random.randn(), name='theta0')
theta1 = tf.Variable(np.random.randn(), name='theta1')
```

TF variable:
A tensor whose value can be modified by running operations on it.

Linear regression

```
# Defining the method to do the minimisation of the cost function
optimiser = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_function)
```

Gradient Descent Optimiser:
Implements the gradient descent algorithm.

Linear regression

```
init = tf.global_variables_initializer()
```

Needs to be run in the session before any other operation!

Variable initialiser:
Operation that assigns a value to the variables in a session.

Exercise

Follow the example of the linear regression to generate a quadratic fit.

The model that you need to use has this formula:

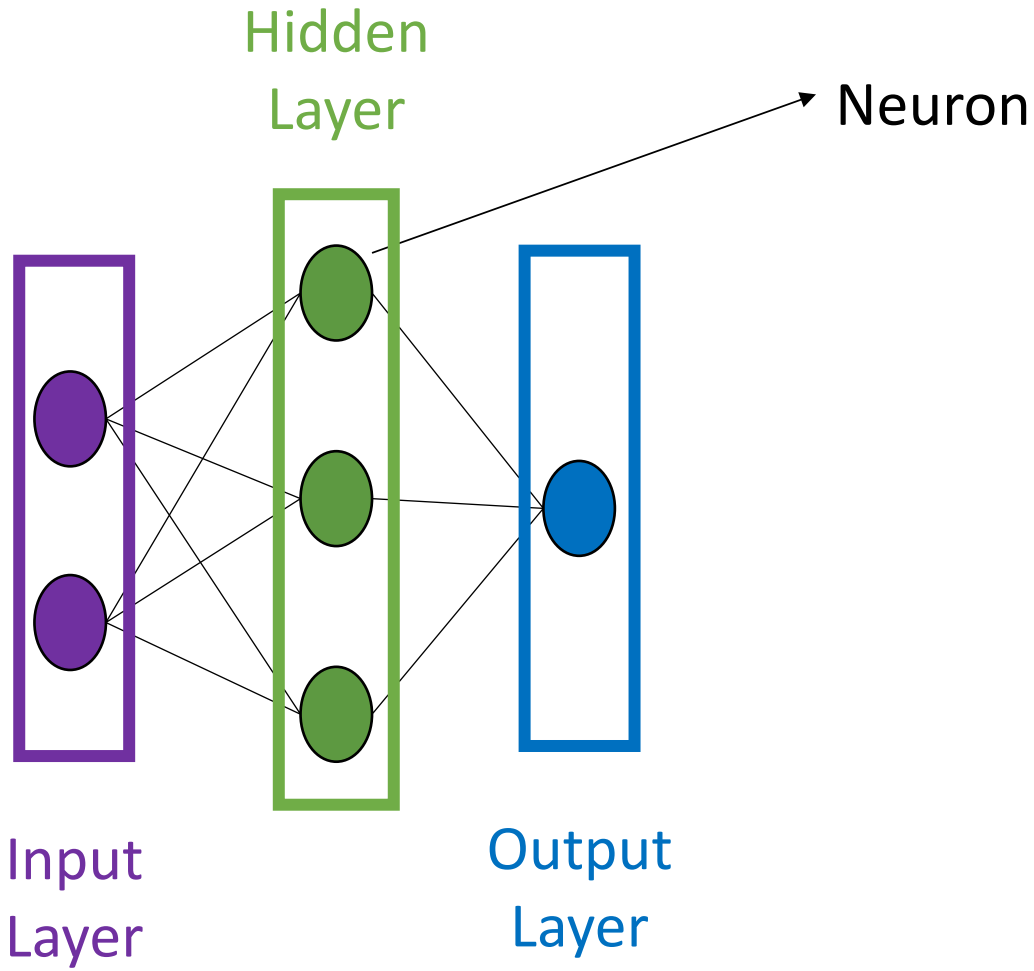
$$y = ax^2 + bx + c$$

Introduction to TensorFlow

Hackathon 2018

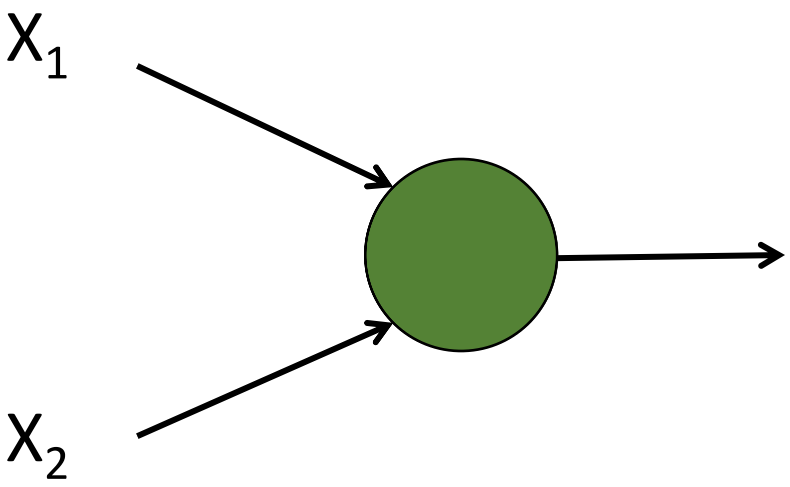
Part 3 – Neural Network!

Neural networks



A tiny neural network

Hidden neuron



Parameters

$$f(w_1x_1 + w_2x_2 + b)$$

For example:

$$f(z) = \frac{1}{1 + e^{-z}}$$

Linear Algebra

To go from the input layer to the output layer:

x_1	x_2

$W_{11}^{(1)}$	$W_{12}^{(1)}$
$W_{21}^{(1)}$	$W_{22}^{(1)}$
$W_{31}^{(1)}$	$W_{32}^{(1)}$

Linear algebra

$$\mathbf{x}\,\mathbf{W}^{\mathrm{T}}$$

x_1	x_2

$W_{11}^{(1)}$	$W_{21}^{(1)}$	$W_{31}^{(1)}$
$W_{12}^{(1)}$	$W_{22}^{(1)}$	$W_{32}^{(1)}$

=

$W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2$	$W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2$	$W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2$

Linear Algebra

$\mathbf{W}^{(l)}$

Matrix of shape N x M

Number of neurons in layer l+1

Number of neurons in layer l

Linear Algebra

$\mathbf{b}^{(l)}$

Matrix of shape 1 x N

Number of
neurons in layer
l+1

Linear Algebra

$$\mathbf{x}\,\mathbf{W}^{\mathrm{T}}$$

x_1	x_2

$\vdots$

$W_{11}^{(1)}$	$W_{21}^{(1)}$	$W_{31}^{(1)}$
$W_{12}^{(1)}$	$W_{22}^{(1)}$	$W_{32}^{(1)}$

$=$

$W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2$	$W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2$	$W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2$

$\vdots$

Training the network

Cost function:

Output of the neural network

$$cost = \frac{1}{N}\sum_{i=1}^{N}\left(Y_N^{Neural\ net} - Y_N^{Data}\right)^2$$

Data point

Number of
data points

Exercise

Modify the neural network so that there are 2 hidden layers instead of one.

Introduction to TensorFlow

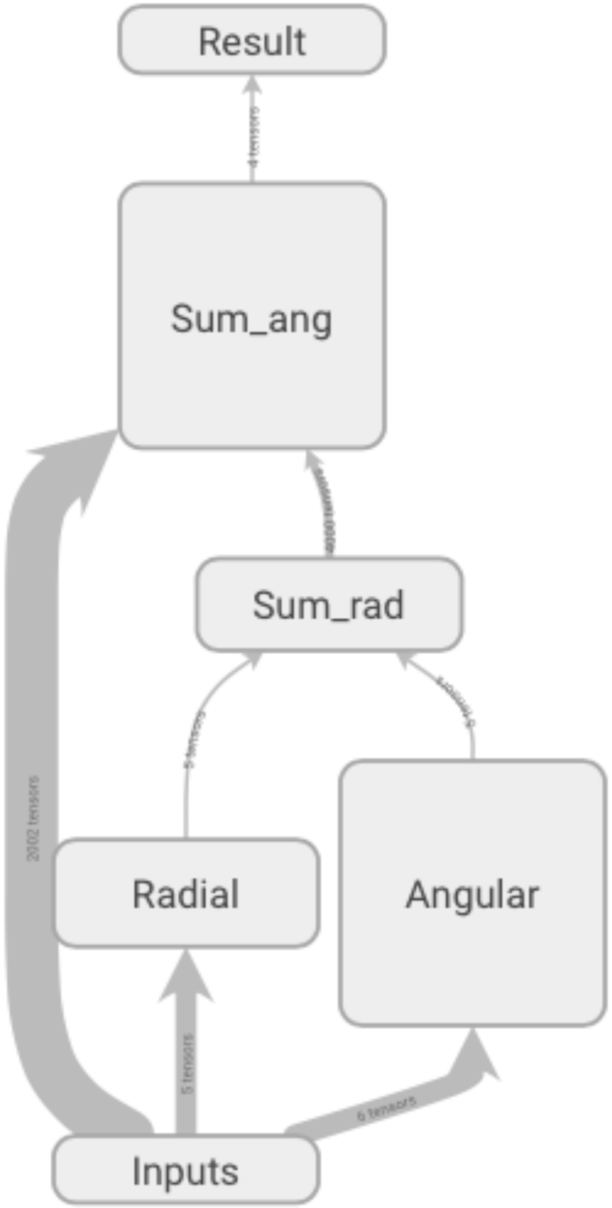
Hackathon 2018

Part 4 – Extra stuff

Tensorboard

Web interface for graph visualization and manipulation

- Group nodes in “Name scopes”
- (Add scalar and histogram summaries to your operations)
- Instantiate a SummaryWriter object
- In a session, pass the summaries to the SummaryWriter



Tensorboard

- After training, a directory is created containing:

`events.out.tfevents.1524151475.Silvias-MacBook-Pro.local`

- On command line, travel to that directory
- Type:

```
tensorboard --logdir=.
```

- Copy the address that comes out into a browser:

`http://Silvias-MBP:6006`

Dataset

tf.data
API that lets you build complex input pipelines

- **tf.data.Dataset**
- **tf.data.Iterator**

Dataset

```
with tf.name_scope("Data"):
    x_ph = tf.placeholder(dtype=tf.float32, shape=[None, 1])
    y_ph = tf.placeholder(dtype=tf.float32, shape=[None, 1])

    dataset = tf.data.Dataset.from_tensor_slices((x_ph, y_ph))
    dataset = dataset.batch(batch_size)
    iterator = tf.data.Iterator.from_structure(dataset.output_types, dataset.output_shapes)
    tf_x, tf_y = iterator.get_next()
```

Create dataset from tensors or from TFRecord files

Dataset

```
with tf.name_scope("Data"):
    x_ph = tf.placeholder(dtype=tf.float32, shape=[None, 1])
    y_ph = tf.placeholder(dtype=tf.float32, shape=[None, 1])

    dataset = tf.data.Dataset.from_tensor_slices((x_ph, y_ph))
    dataset = dataset.batch(batch_size)
    iterator = tf.data.Iterator.from_structure(dataset.output_types, dataset.output_shapes)
    tf_x, tf_y = iterator.get_next()
```

Extract elements from the data set using the iterator

Note: when you run an operation that depends on tf_x and tf_y, it will automatically run the iterator.get_next() operation.

Dataset

```
# Initialisation of the model
init = tf.global_variables_initializer()
iterator_init = iterator.make_initializer(dataset)

training_cost = []

# Running the graph
with tf.Session() as sess:
    sess.run(init)
    sess.run(iterator_init, feed_dict={x_ph:x_col, y_ph:y_col})
```

Creating an initialisation operation

Running the initialisation operation

Dataset

```
for i in range(self.iterations):
    # This will be used to calculate the average cost per iteration
    avg_cost = 0
    # Learning over the batches of data
    for j in range(n_batches):
        batch_x = x[indices][j * batch_size:(j+1) * batch_size]
        batch_y = y[indices][j * batch_size:(j+1) * batch_size]
        feed_dict = {tf_x: batch_x, tf_y: batch_y}
        opt, c = self.session.run([optimizer, cost], feed_dict=feed_dict)
        avg_cost += c * batch_x.shape[0] / x.shape[0]
```

Dataset

```
for iter in range(iterations):
    sess.run(iterator_init, feed_dict={x_ph: x_col, y_ph: y_col})
    for batch in range(n_batches):
        opt, c = sess.run([optimizer, cost])
```