Gentlest Intro to Tensorflow

Khor, @neth_6, re:Culture

In collaboration with (Sam & Edmund):

AheadConsulting

Goal

- Perform **linear regression** in its most basic form through simplest example
  - Predict house price with just single feature, i.e., house size

What We Will Learn Together

- 4 steps of Machine Learning:
  - Create model
  - Define cost function
  - Collect data
  - Train
- 2 parts of Tensorflow:
  - Model linear regression & cost function as Tensorflow graph
  - Train Tensorflow graph with dataset

Machine Learning

Machine Learning Purpose

Predict something given a set of **features** that influences prediction:

- Predict house price given its size, location, configuration (features)

Machine Learning Training

Before predicting, we have to:

- Choose/Create a **model**
- **Train** the model to learn prediction with **data**

Problem

Predict House Price Given House Size

y = house price

x = house size (feature)

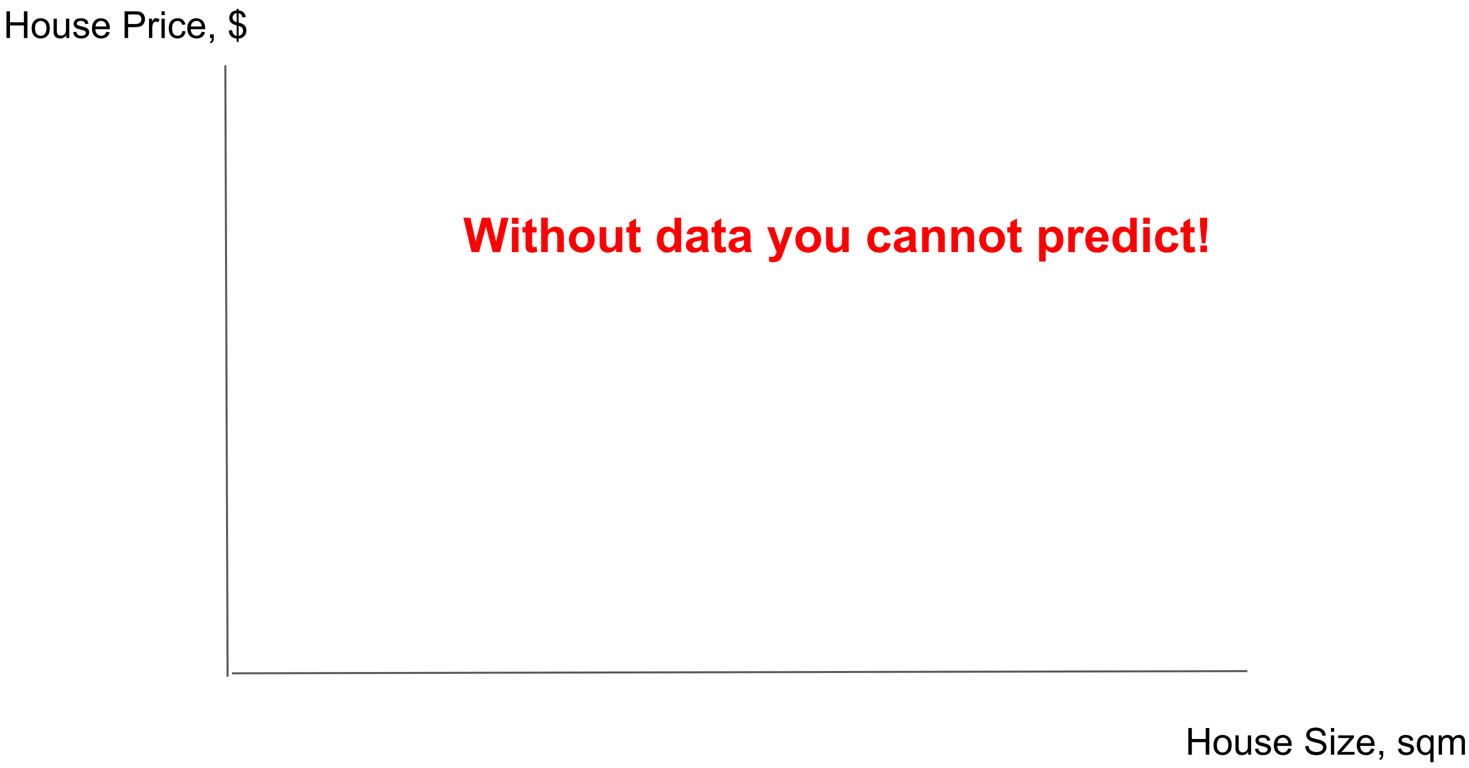
Solve:

- Given a house size, what should be the house price?

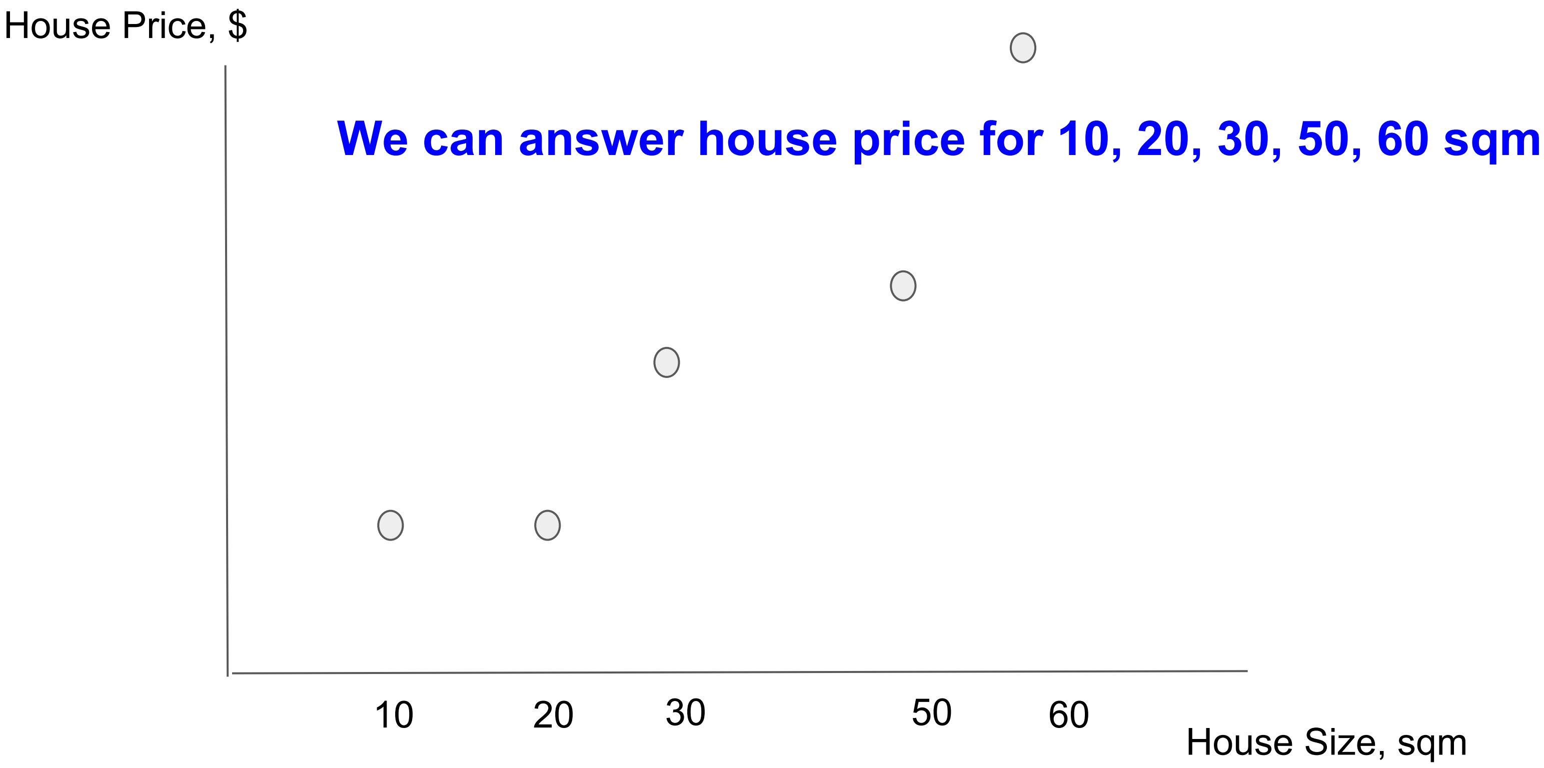
Model

What is the House Price?

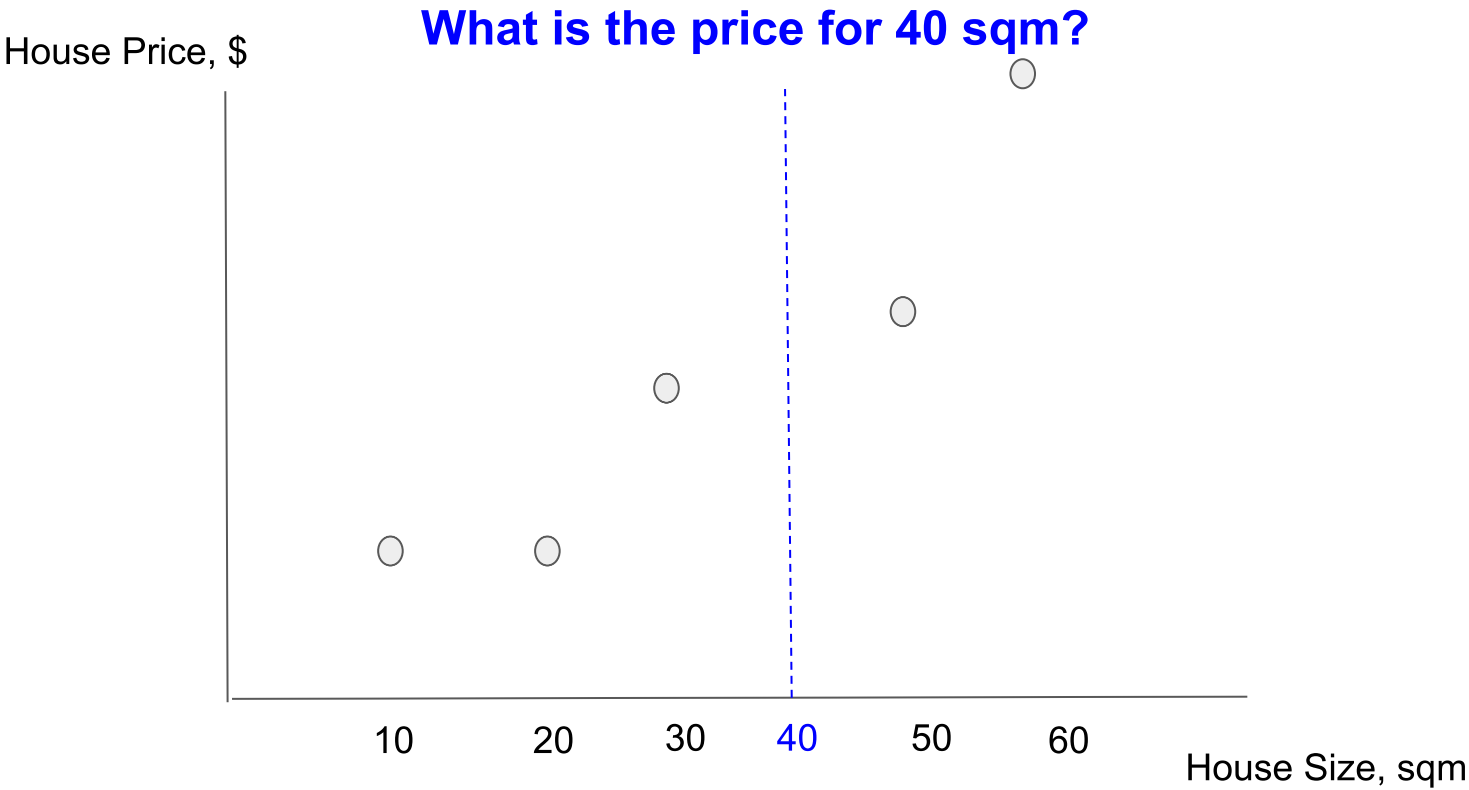
What is the House Price?



What is the House Price?

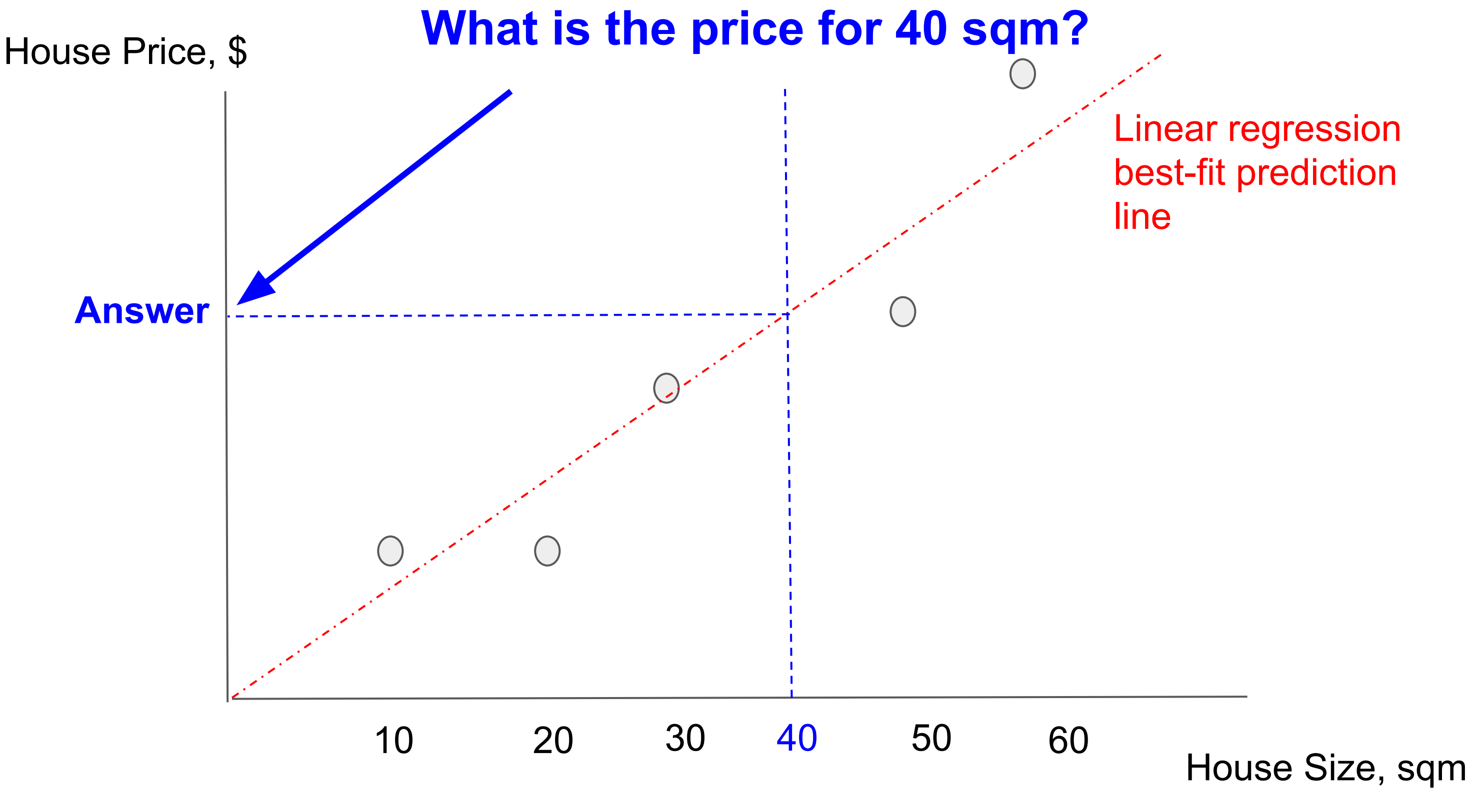


What is the House Price?

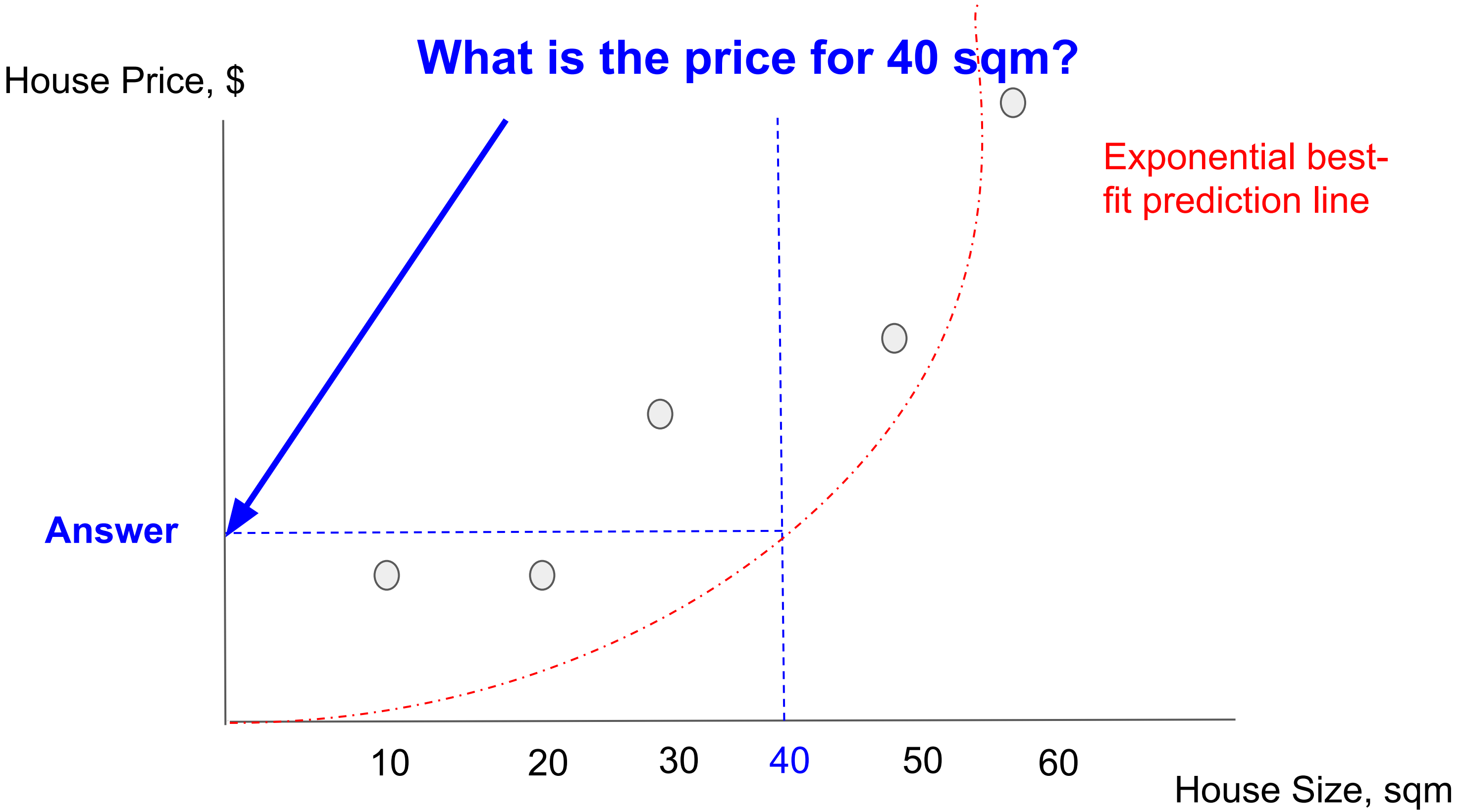


What is the House Price?

What is the House Price?



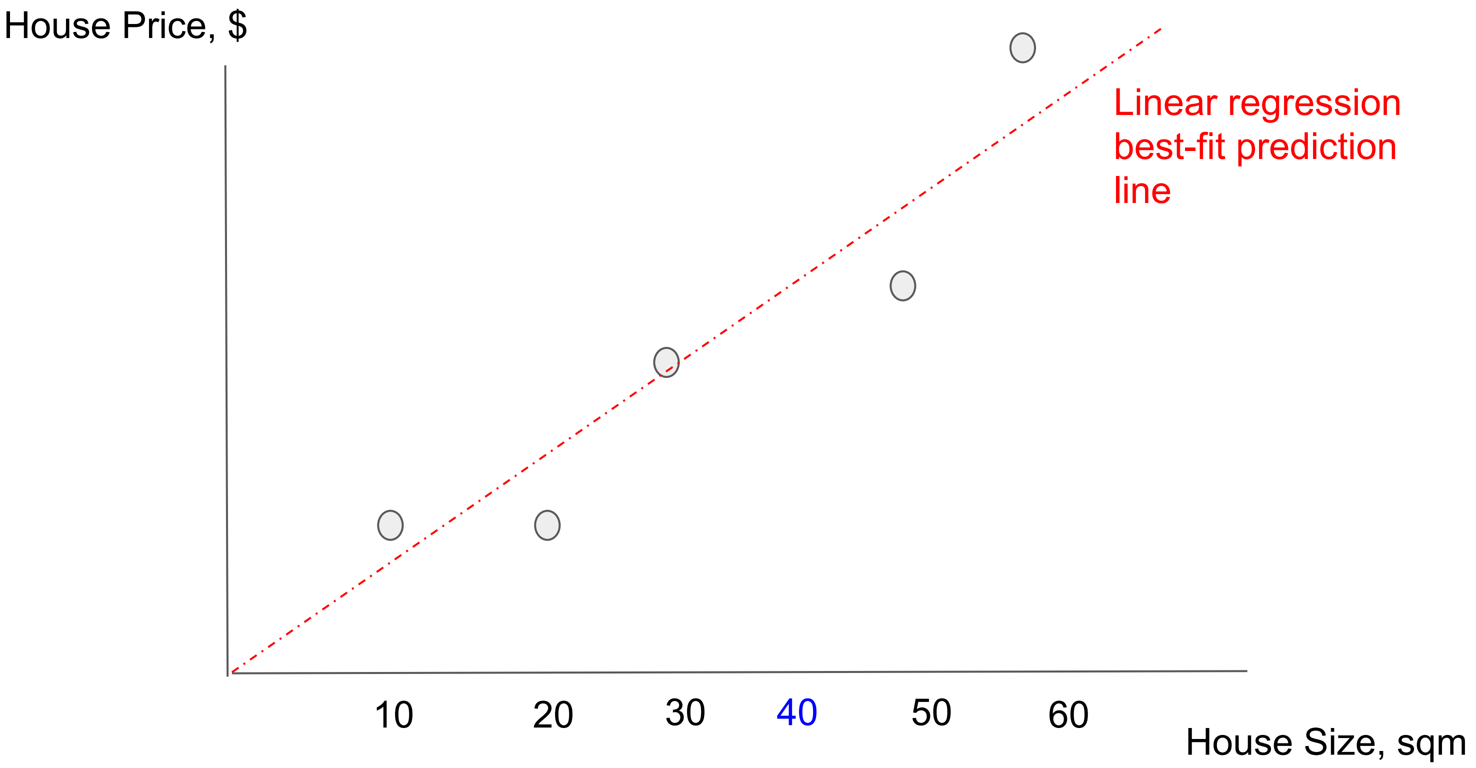
What is the House Price?



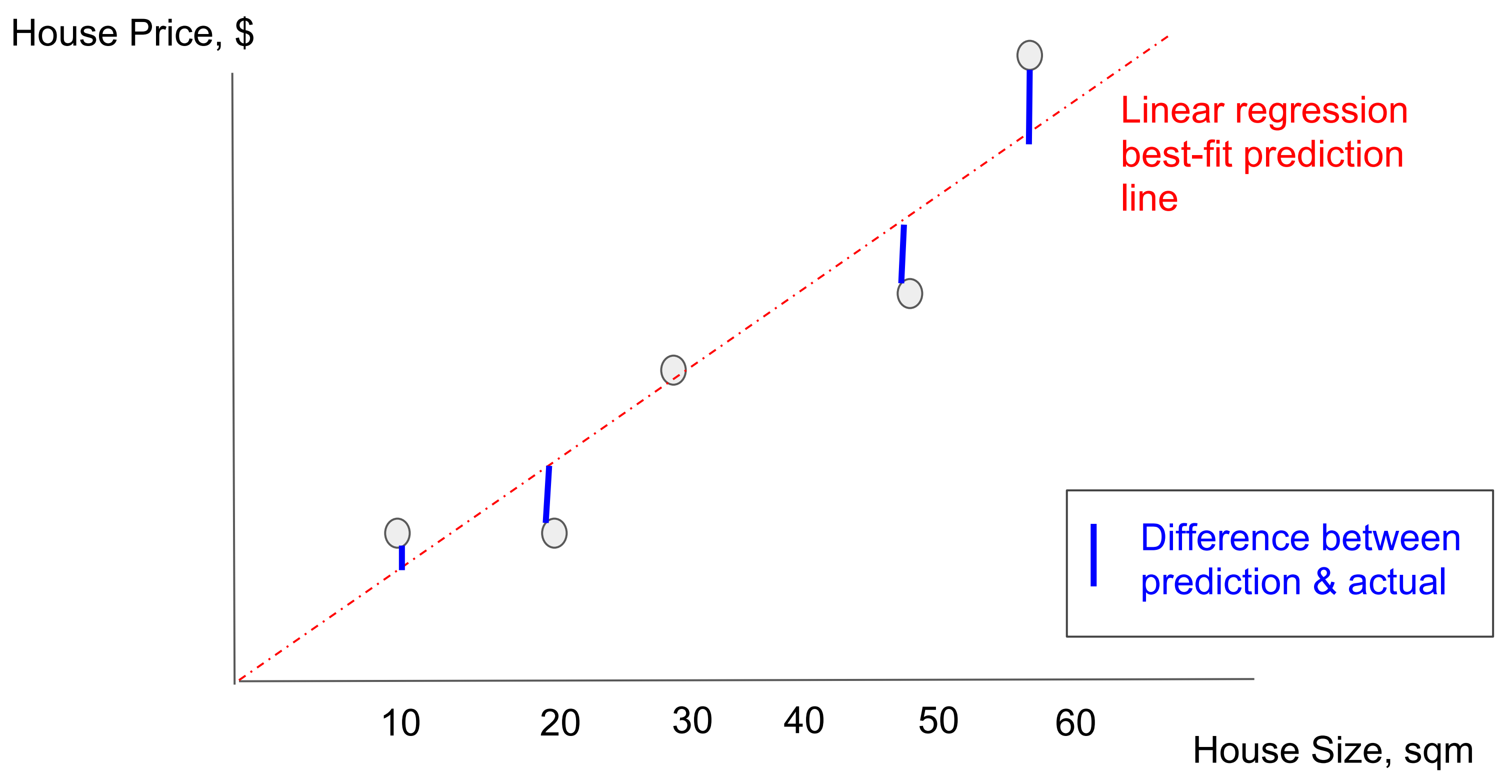
Cost Function: Best-fit Prediction

Minimize difference between predicted & actual values

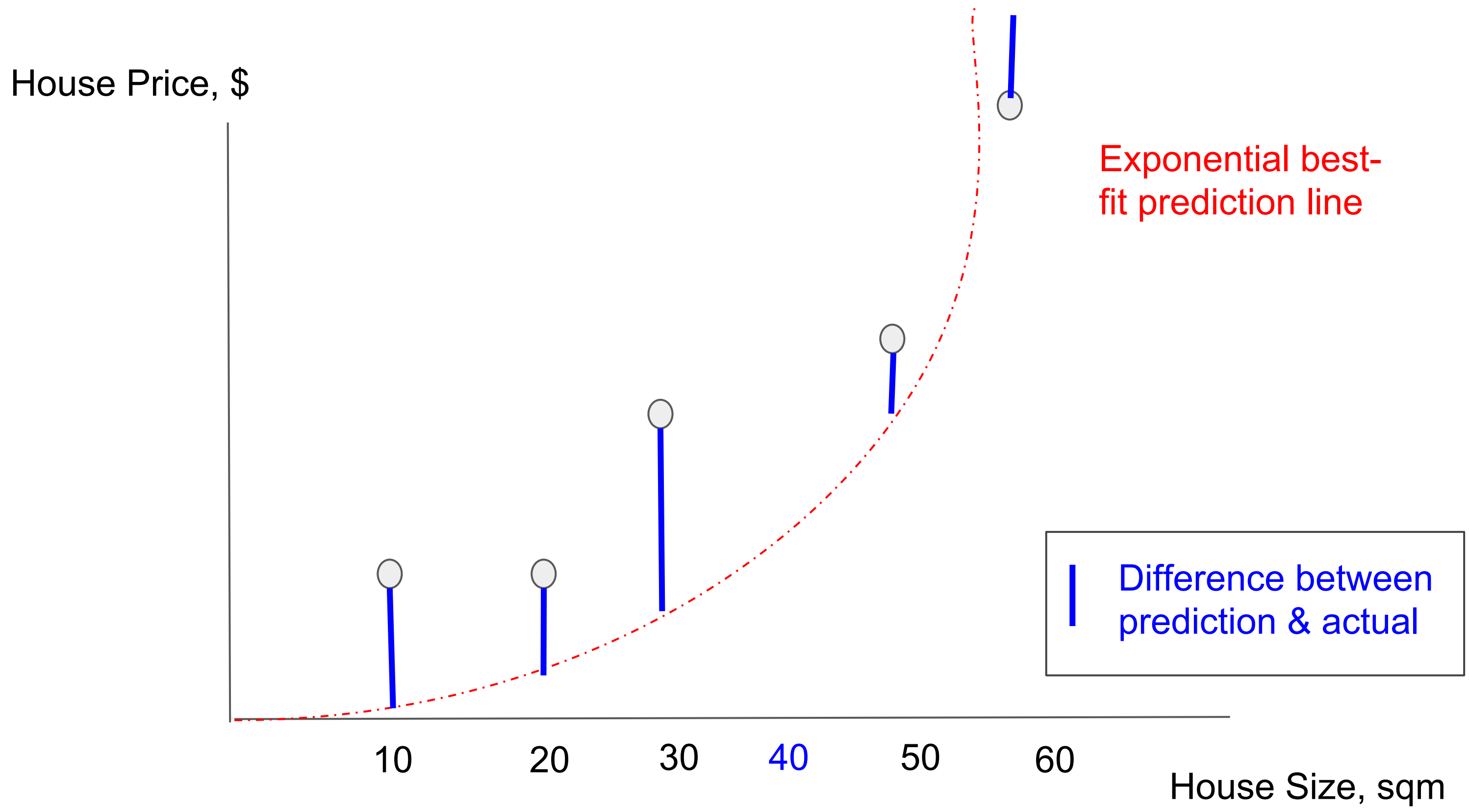
What is the House Price?



What is the House Price?



What is the House Price?



Cost function: How different is your model from reality

sum((y_ - y)**2)

Where:

- y_: Actual value
- y: Predicted value
- **2: Power of 2

Linear Regression

Linear Regression

$$y = W.x + b$$

y: Predicted house price

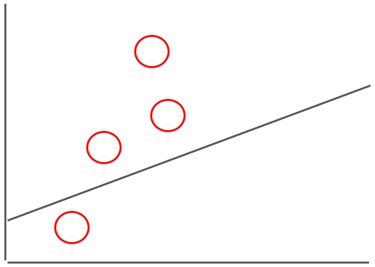
x: House size from dataset

Find a good W, b

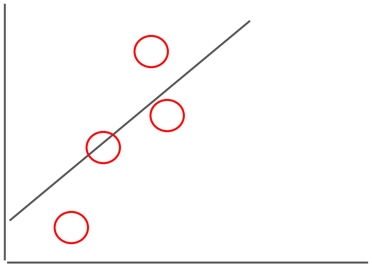
W: Gradient (Steepness)

$$W1 < W2 < W3$$

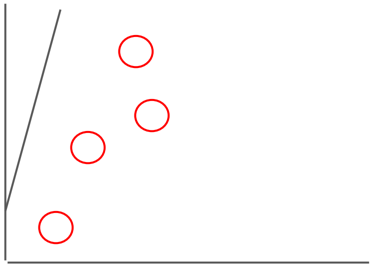
$y = W1.x + b$



$y = W2.x + b$



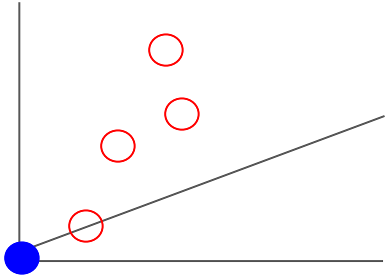
$y = W3.x + b$



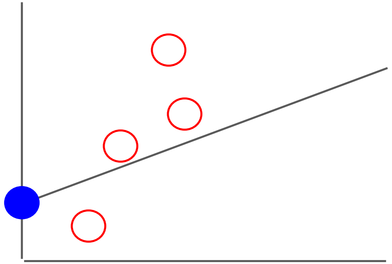
b: Intersect

$b1 < b2 < b3$

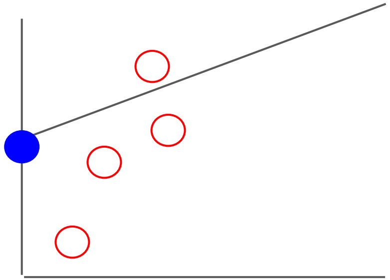
$y = W.x + b1$



$y = W.x + b2$



$y = W.x + b3$



Train

Train: Find W, b

```
W_best, b_best = 0, 0
Loop J times:
  Choose current_W, current_b
  If cost(current_W, current_b) < cost(W_best, b_best):
    W_best, b_best = current_W, current_b
```

Train: Find W, b

W_best, b_best = 0, 0

Loop J times:

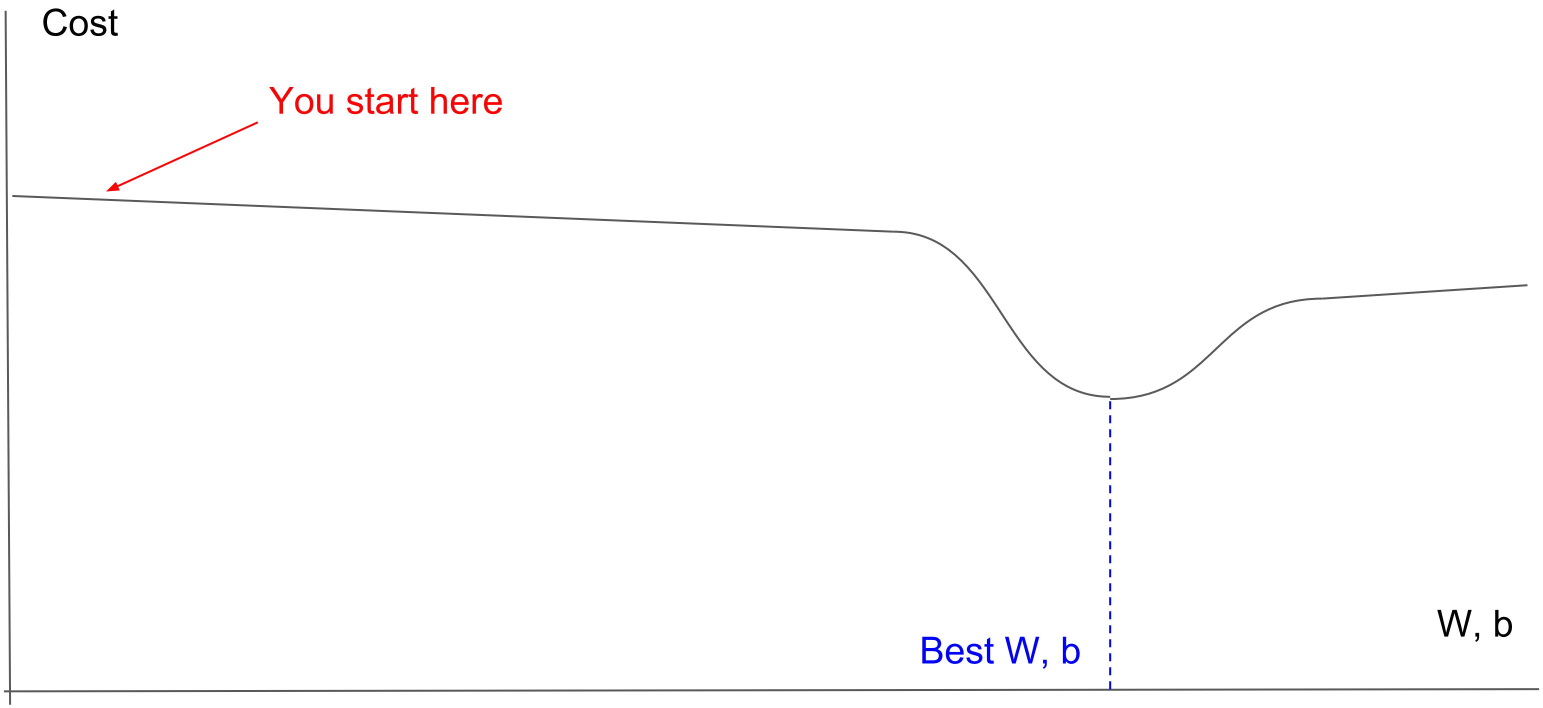
  Choose current_W, current_b

  If cost(current_W, current_b) < cost(W_best, b_best):

    W_best, b_best = current_W, current_b

What is the best way to choose current_W, current_b?

Finding Minimum Cost Function



How to Reach Minimum?

Choosing Training Method

- Random
  - Could take forever
- Gradient descent
  - From current viewpoint, move towards direction where steepness is greatest

Ready?

Recap:

- Model: Linear regression, y = W.x + b
- Cost: Least squared, cost = sum((y_ - y)**2)
- Train: Gradient descent

Step 1: Model $y = W.x + b$

Tensorflow Model

tf.placeholder

- Hold data to be feed into the model
- x = tf.placeholder(tf.float32, [None, 1])

tf.Variable

One output, house price

One feature, house size

- Hold variables to be trained
- W = tf.Variable(tf.zeros([1,1]))
- b = tf.Variable(tf.zeros([1]))

Tensorflow Model

tf.matmul(x,W)

- Multiply 2 matrices
- product = tf.matmul(x,W)

‘+’

- y = product + b
- Expands to:
  - y = tf.matmul(x,W) + b = W.x + b **Done!**

Step 2: Cost Function sum((y_ - y)** 2)

Best-fit: Minimize Difference Prediction and Actual

Actual values:

One output, house price

- y_ = tf.placeholder(tf.float32, [None, 1])

Minimize difference between actual and prediction:

- cost = tf.reduce_sum(tf.pow((y_-y), 2))

Step 3: Data

Faking Data

```
for i in range(100):
  // Create fake data for actual data
  xs = np.array([[i]])
  ys = np.array([[2*i]])
```

- xs = 0, 1, 2, …., 99
- ys = 0, 2, 4, …., 198
- ys is always twice of xs
- **y = W.x + b where W = 2, b = 0 => Best fit: y = 2x**

Step 4: Train

Train using Gradient Descent

Train using Gradient Descent with steps in 0.000001

- train_step = tf.train.GradientDescentOptimizer(0.00001).minimize(cost)

Train Model

- sess = tf.Session()
- init = tf.initialize_all_variables()
- sess.run(init)
- steps = 100

Tensorflow Non-interactive Mode

Nothing is actually executed until sess.run(...)

Go!!!!!

```
for i in range(steps):
  # Create fake data for y = W.x + b where W = 2, b = 0
  xs = np.array([[i]])
  ys = np.array([[2*i]])

  # Train
  feed = { x: xs, y_: ys }
  sess.run(train_step, feed_dict=feed)

  print("After %d iteration:" % i)
  print("W: %f" % sess.run(W))
  print("b: %f" % sess.run(b))
```

Result

Output

After 0 iteration:

W: 0.000000

b: 0.000000

After 1 iteration:

W: 0.000040

b: 0.000040

…

…

After 98 iteration:

W: 1.997181

b: 0.051121

After 99 iteration:

W: 1.997632

b: 0.051126

After 100 iterations:
W ~ 2.0 and b ~ 0.0

Debugging

When Gradient Descent Go Awry?

Re-run:

```
sess.run(tf.initialize_all_variables())
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
for i in range(steps):
  ....
```

Output

After 0 iteration:

W: 0.000000

b: 0.000000

After 1 iteration:

W: 0.040000

b: 0.040000

After 2 iteration:

W: 0.195200

b: 0.117600

...

...

After 46 iteration:

W: 323066974878921632721935905090174976.000000

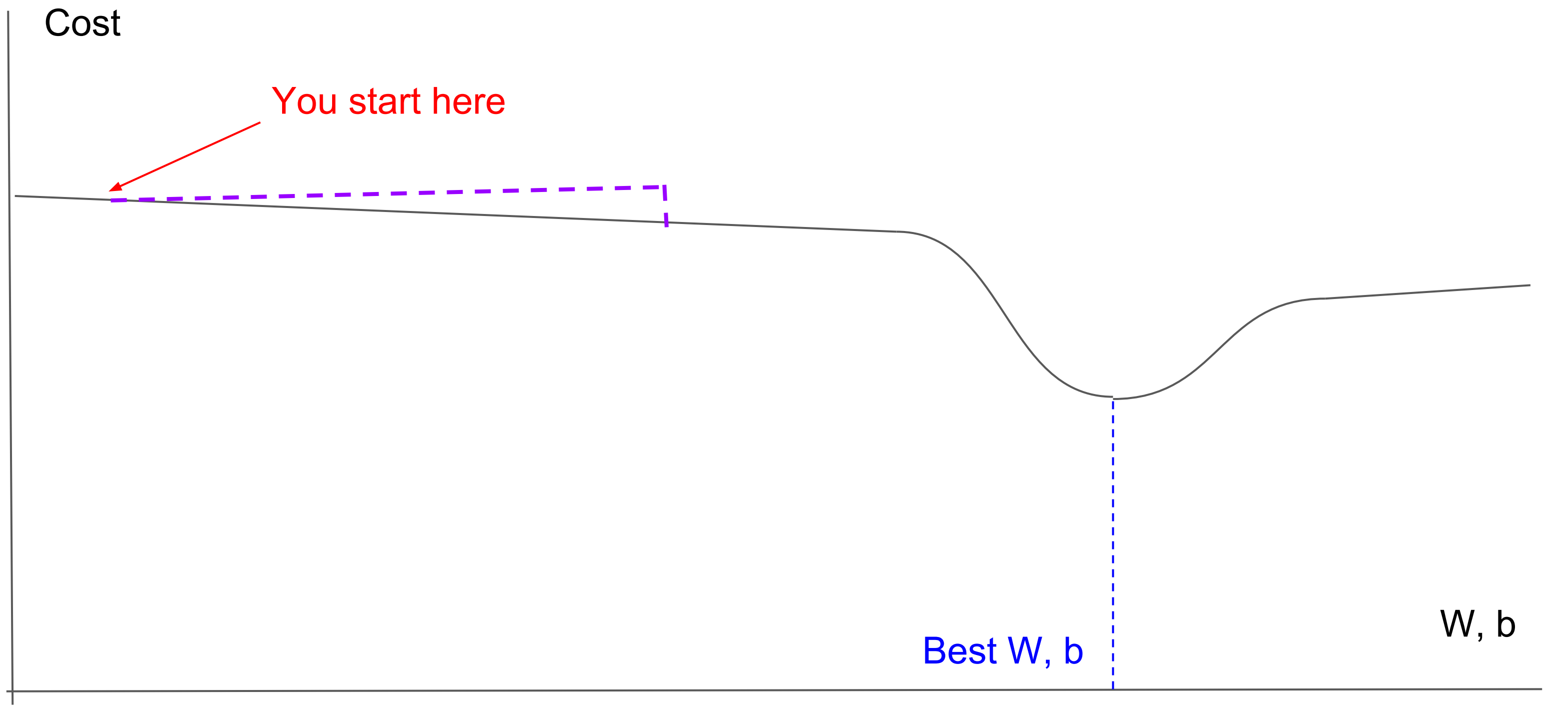
b: 701951740150071676574627696895590 4.000000

After 47 iteration:

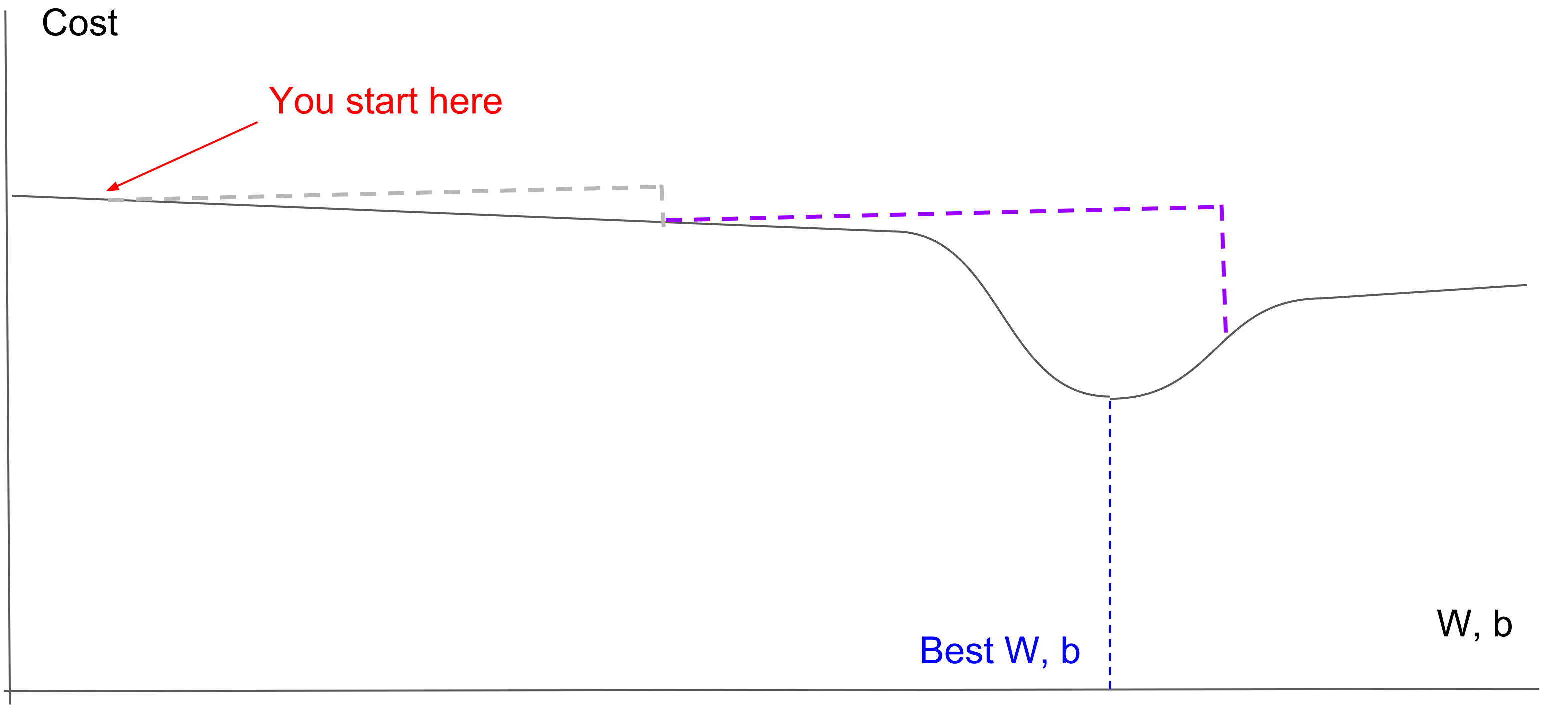
W: -inf

b: -2968038293574744331138960048531701 76.000000

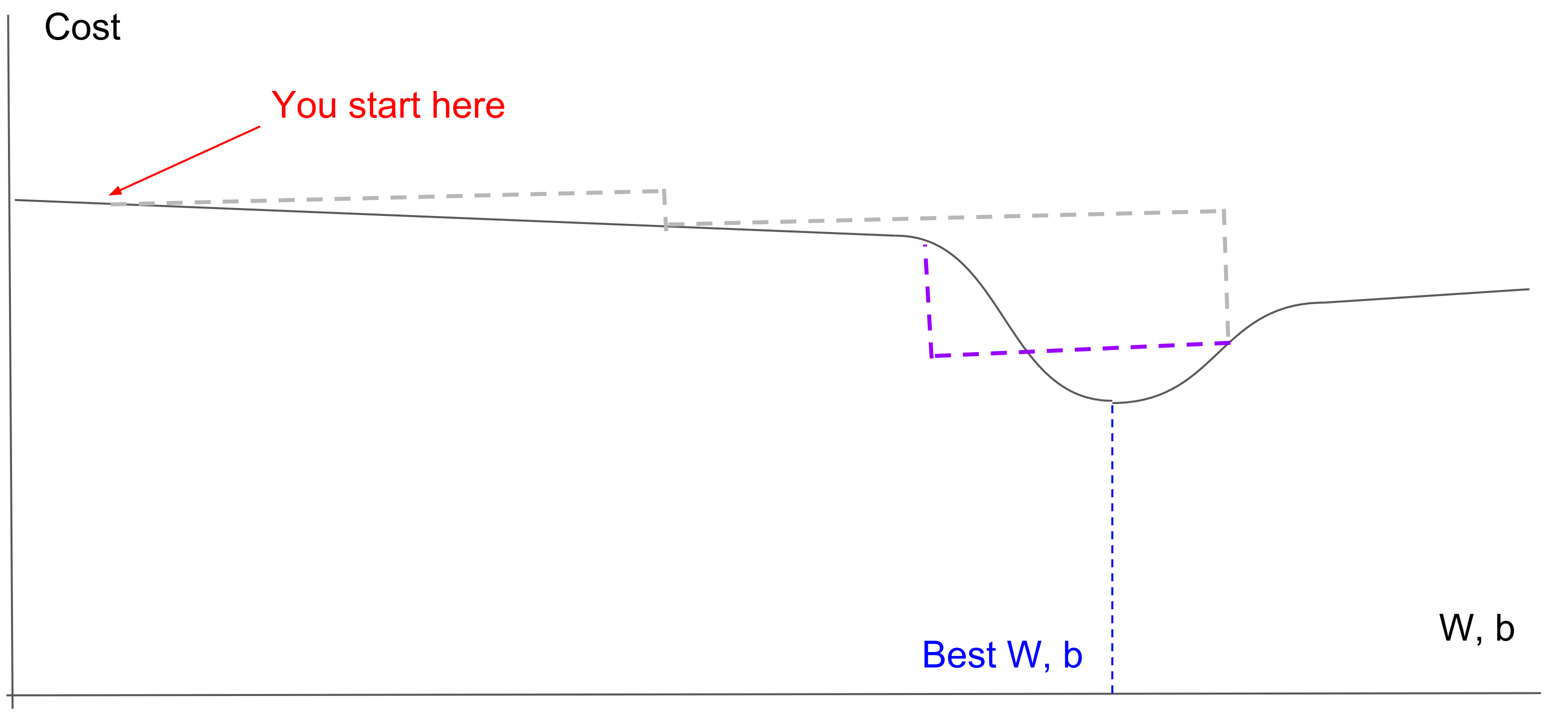
Finding Minimum Cost Function



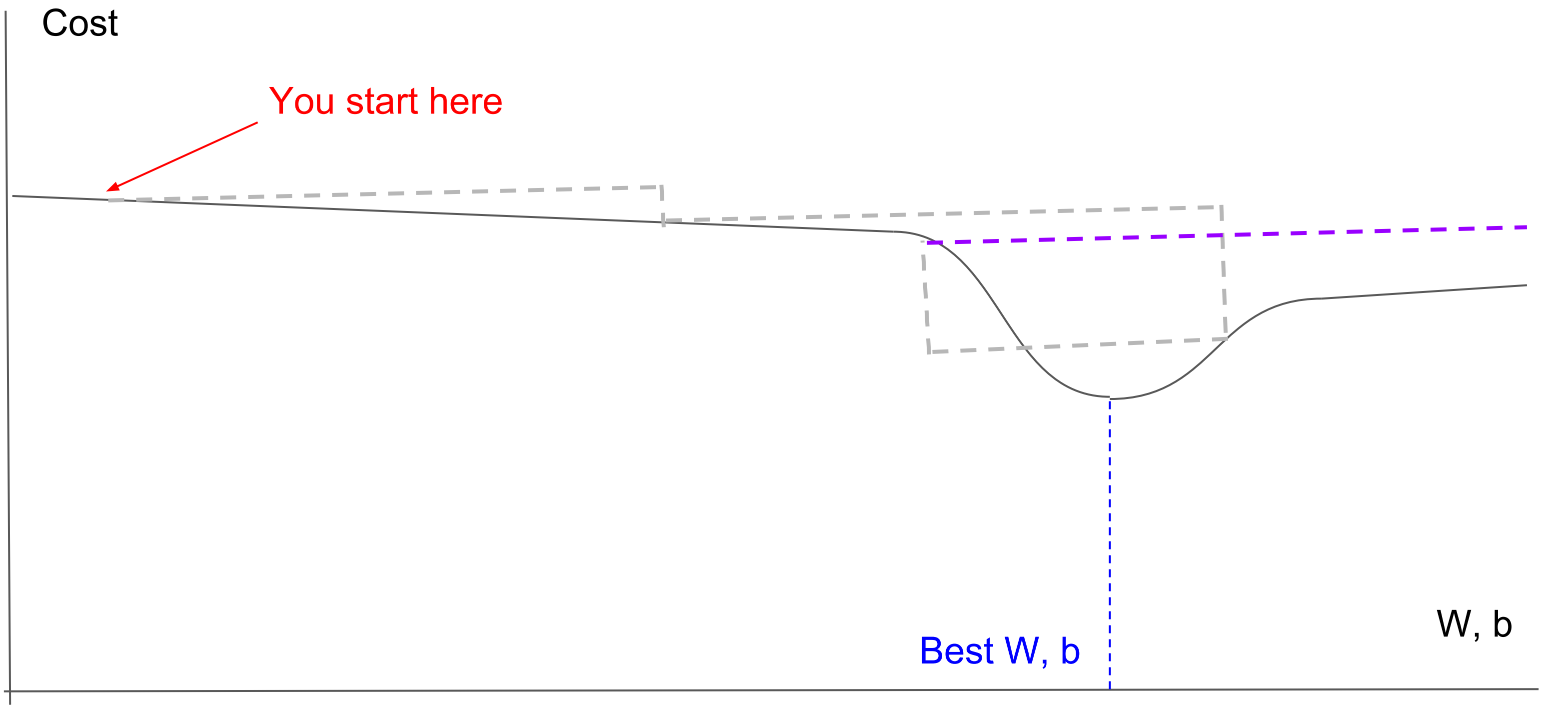
Finding Minimum Cost Function



Finding Minimum Cost Function



Finding Minimum Cost Function



Errors: Missing Placeholders

feed = { x: xs, y: ys }

sess.run(train_step, feed_dict=feed)

- y should be y_

W tensorflow/core/common_runtime/executor.cc:1076] 0x229e510 Compute status: Invalid argument: You must feed a value for placeholder tensor 'Placeholder_1' with dtype float

[[Node: Placeholder_1 = Placeholder[dtype=DT_FLOAT, shape=[], _device="/job:localhost/replica:0/task:0/cpu:0"]()]]

Placeholder Numbers

```
cost = tf.reduce_sum(tf.pow((y_-y), 2))
```

```
y_ = tf.placeholder(tf.float32, [None, 1])
```

Placeholder_1

```
y = product + b
```

```
product = tf.matmul(x,W)
```

```
x = tf.placeholder(tf.float32, [None, 1])
```

Placeholder_0

Assigning Names to Placeholders

cost = tf.reduce_sum(tf.pow((y_-y), 2))

y_ = tf.placeholder(tf.float32, [None, 1],
name='y-input')

Placeholder_1

y = product + b

product = tf.matmul(x,W)

x = tf.placeholder(tf.float32, [None, 1])

Placeholder_0

Errors: Wrong Shape

```
for i in range(steps):
  xs = np.array([i])
  ys = np.array([[2*i]])
```

- xs should be np.array([[i]])

Traceback (most recent call last):
File "<stdin>", line 7, in <module>
File "/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/session.py", line 364, in run
tuple(subfeed_t.get_shape().dims)))
ValueError: Cannot feed value of shape (1,) for Tensor u'Placeholder:0', which has shape (Dimension(None), Dimension(1))

Placeholder Numbers

```
cost = tf.reduce_sum(tf.pow((y_-y), 2))
```

```
y_ = tf.placeholder(tf.float32, [None, 1])
```

Placeholder_1

```
y = product + b

product = tf.matmul(x,W)

x = tf.placeholder(tf.float32, [None, 1])
```

Placeholder_0

Print: Simplest Debugger

print (sess.run(product), feed_dict={ x: np.array([1]) })

```
  File "<stdin>", line 1
        print (sess.run(product), feed_dict={ x: np.array([1]) })
                                 ^
SyntaxError: invalid syntax
>>> print (sess.run(product, feed_dict={ x: np.array([1]) }))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python2.7/dist-packages/tensorflow/python/client/session.py", line 364, in run
        tuple(subfeed_t.get_shape().dims)))
ValueError: Cannot feed value of shape (1,) for Tensor u'Placeholder:0', which has shape (Dimension(None), Dimension(1))
```

Same error so must be it!

Progressing to Tensorflow Tutorial

Difference

Parameters	Gentlest Tensorflow Tutorial	Tensorflow Tutorial
Purpose	Find house price (Float32)	Classify image
y	Each sample is 1 x 1; Single-outcome (house price). Feature value is Float32.	Each sample is 10 x 1: One-hot vector for 10 possible values. Vector element is binary.
x	Each sample is 1 x 1: Single-feature (house size) 1 x 1. Feature value is Float32.	Each sample is 1 x 784: 784 features (28 x 28 bits for each image). Feature value is binary
W	1 x 1: Single-outcome, single-feature	784 x 10: one for each 784 bits and 10 probability its one of 10 possible values

One-hot vector for image 4

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Difference

Parameters	Gentlest Tensorflow Tutorial	Tensorflow Tutorial
b	1 x 1: Bias for single-outcome	10 x 1: Bias for 10 possible values of on-hot vector
cost	Square of difference	Entropy-based
gradient descent batch size	Single sample x, y	100 samples each batch

Summary Machine Learning

4 steps:

- Build **model** (linear regression, logistic regression, neural networks, etc.)
- Define **cost function** to determine how well current model fits actual
- Collect and prepare **data**
- **Train** model to tweak model based on cost function using data

Summary of Tensorflow

- Modelling
  - tf.zeros
  - tf.placeholder
  - tf.Variable
  - tf.matmul
  - tf.reduce_sum
  - tf.pow
  - tf.train.GradientDescentOptimizer(step_size).minimize(cost)
- Train
  - init = tf.initialize_all_variables()
  - sess = tf.session()
  - sess.run(init)
  - sess.run(train, feed_dict={ x: …, y: … })

Questions

- Why not just use numpy.dot methods instead of tf.matmul?
    - tf.matmul is a Tensorflow Operation, and Tensorflow can easily assign each parallelizable operation to an available nodes to speed up operations

References

Code: https://github.com/nethsix/gentle_tensorflow/blob/master/code/linear_regression_one_feature.py