Gentlest Intro to Tensorflow (Part 2)

Khor Soon Hin, @neth_6, re:Culture

In collaboration with Sam & Edmund

Next Steps

- Review of Tensorflow
  - In illustrations
  - tf.placeholder & feed
  - Training process
- Tensorboard: Visualize **cost**, **W**, **b**
- Training batch size: Stochastic/Mini-batch/Full-batch

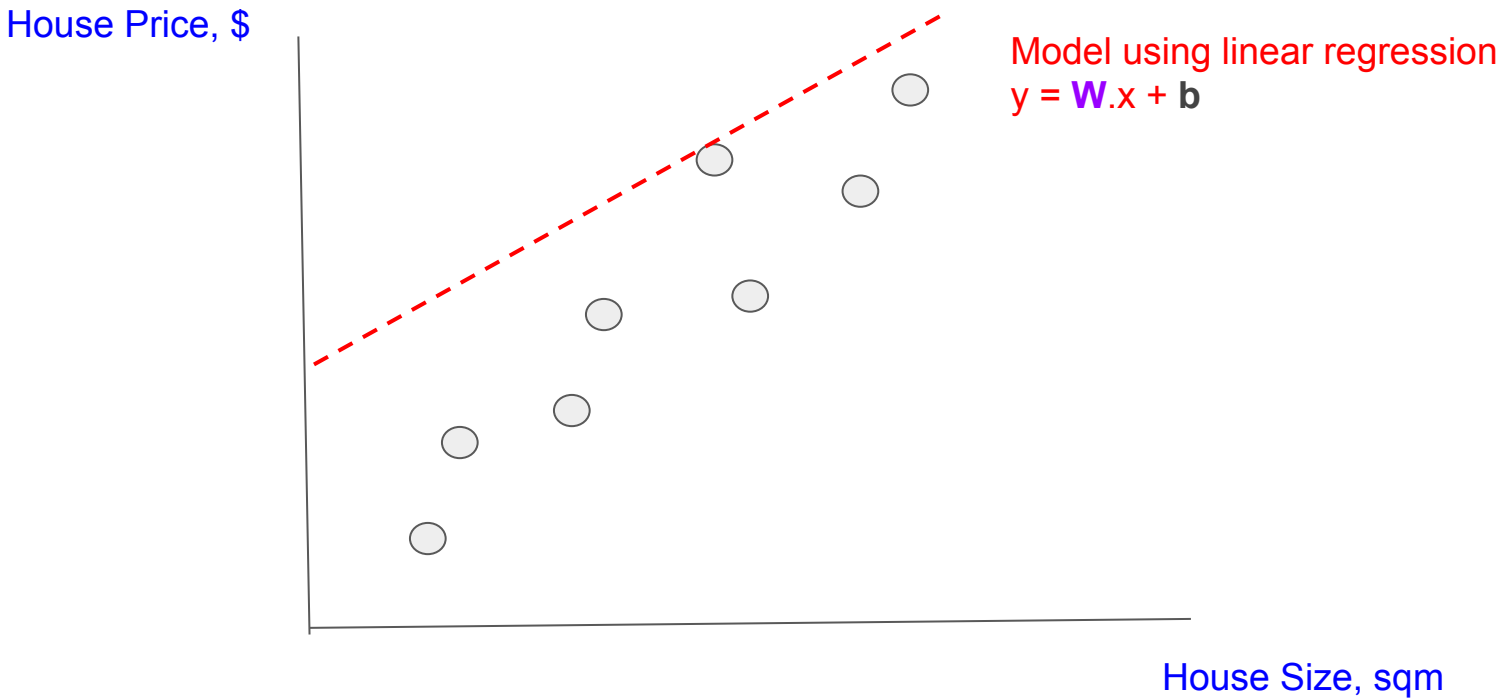
Quick Review

Given house size, use machine learning to predict house price

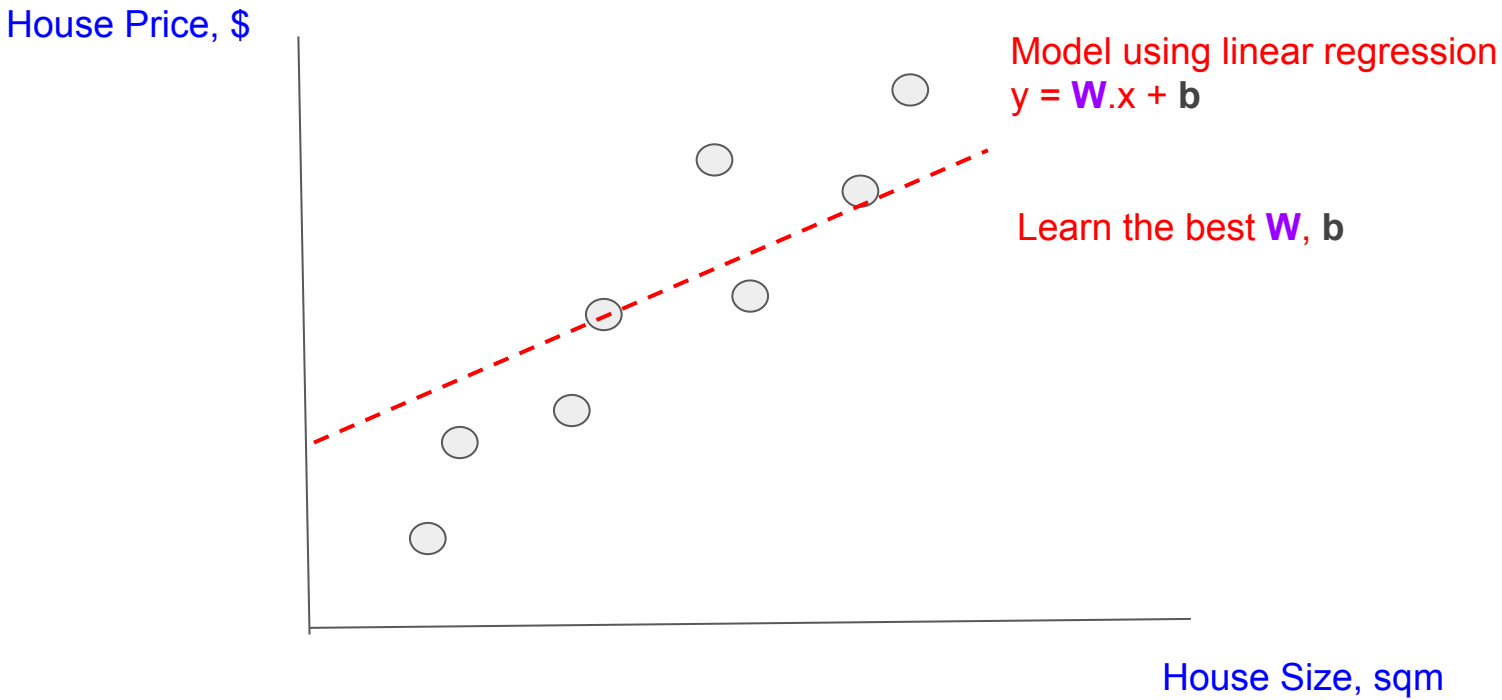
Quick Review

Quick Review

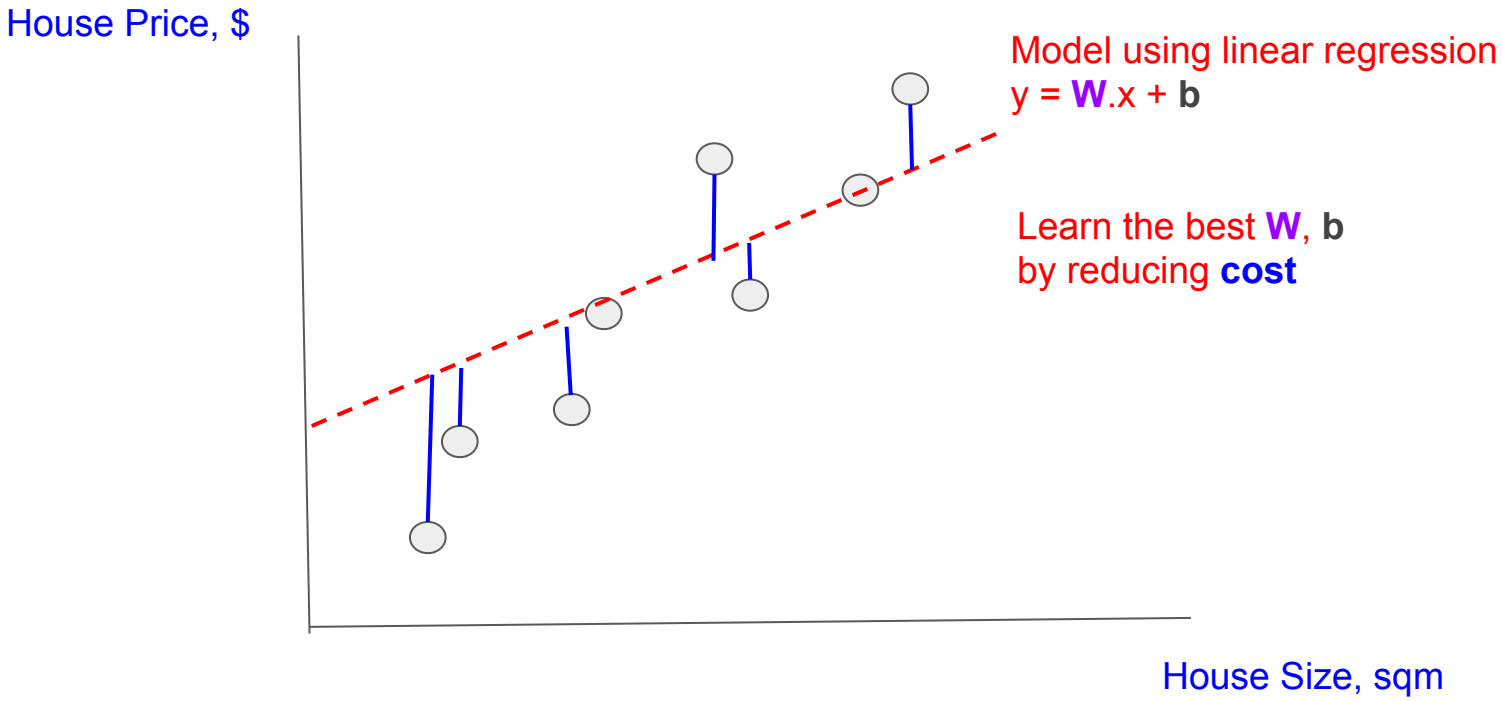
Quick Review



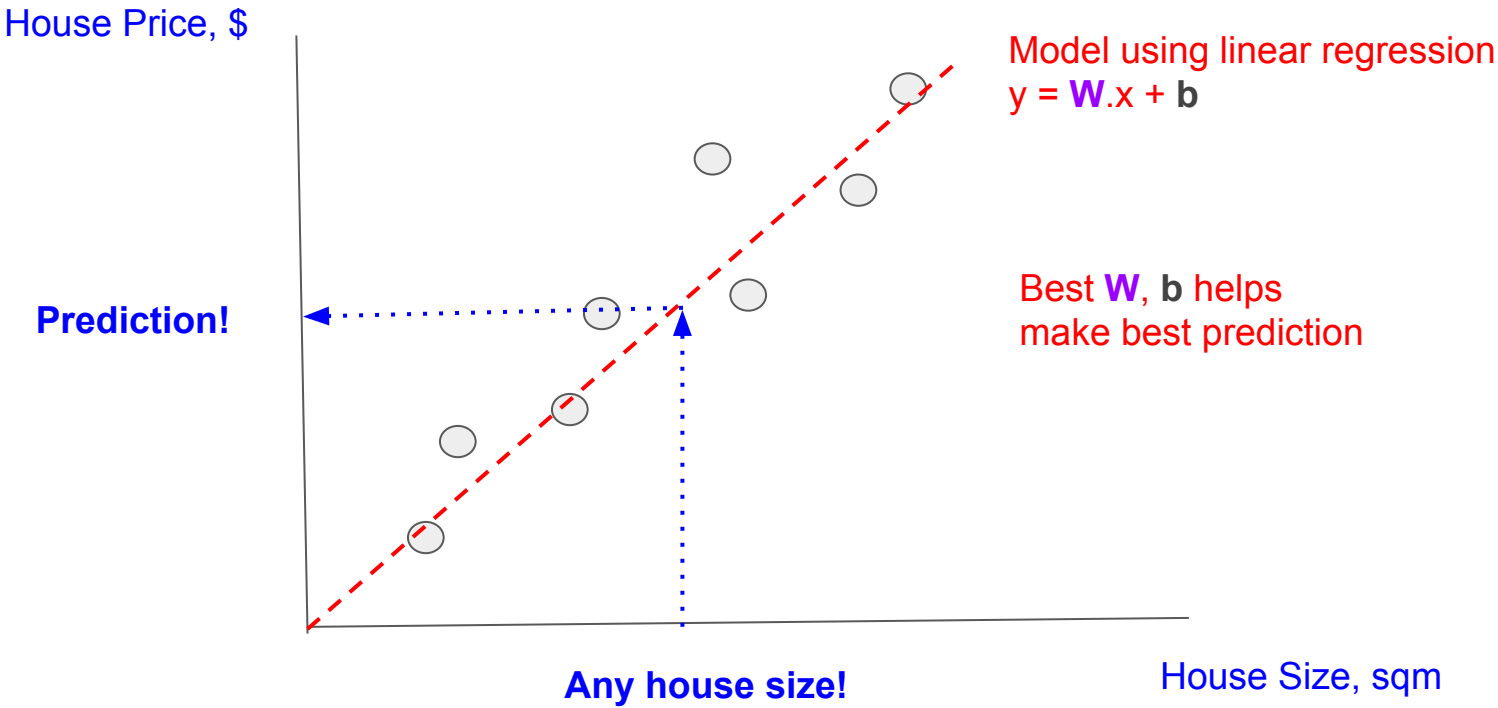
Quick Review



Quick Review



Quick Review



Quick Review

Modeling linear regression in Tensorflow

Tensorflow Graph

Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

Tensorflow Graph

prediction

Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

Tensorflow Graph

prediction

Goal: Train

Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

Tensorflow Graph

prediction

Goal: Train

Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

How: Minimizing actual vs. prediction

Tensorflow Graph

prediction

Goal: Train

Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

How: Minimizing actual vs. prediction

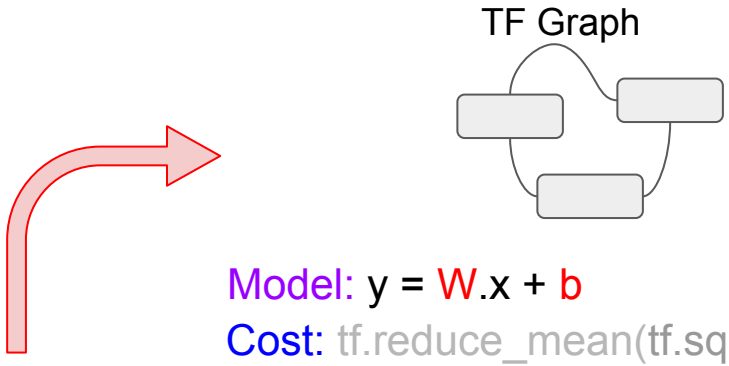
Train: tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

What is Train?

W, b

Init with some values

What is Train?



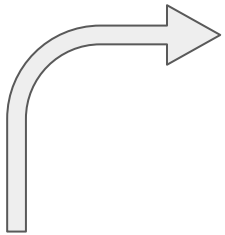
Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

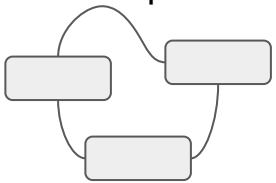
W, b

Init with some values

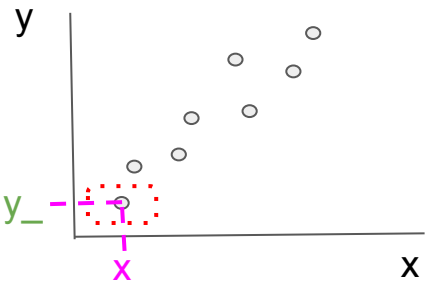
What is Train?



TF Graph



Actual Datapoints



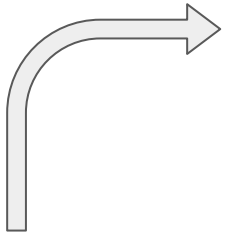
Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

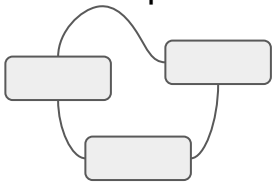
W, b

Init with some values

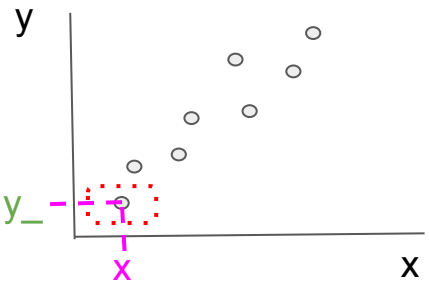
What is Train?



TF Graph



Actual Datapoints

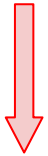


Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

W, b

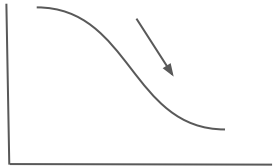
Init with some values



Train: tf.train.GradientDescentOptimizer(01).minimize(cost)

Gradient Descent

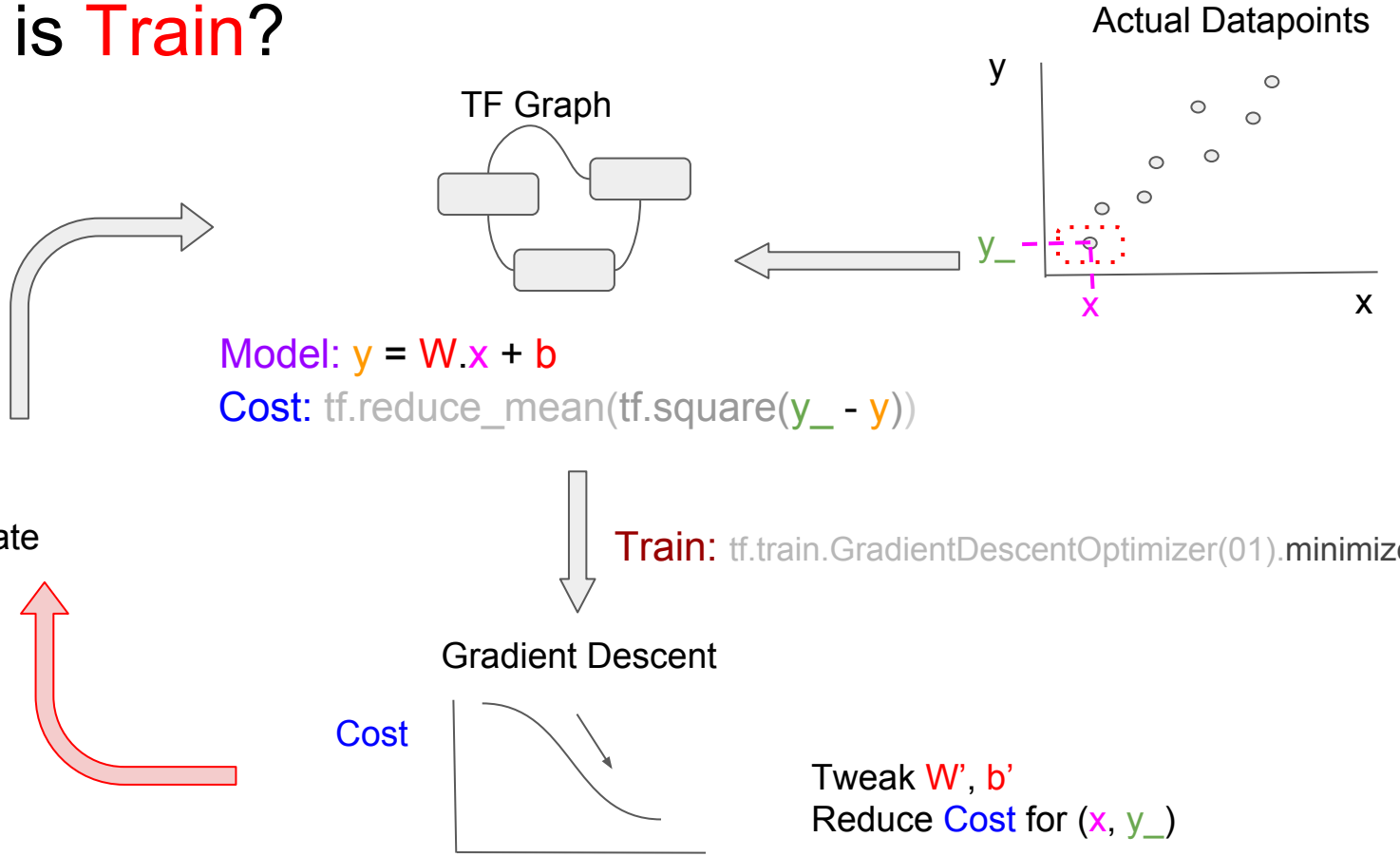
Cost



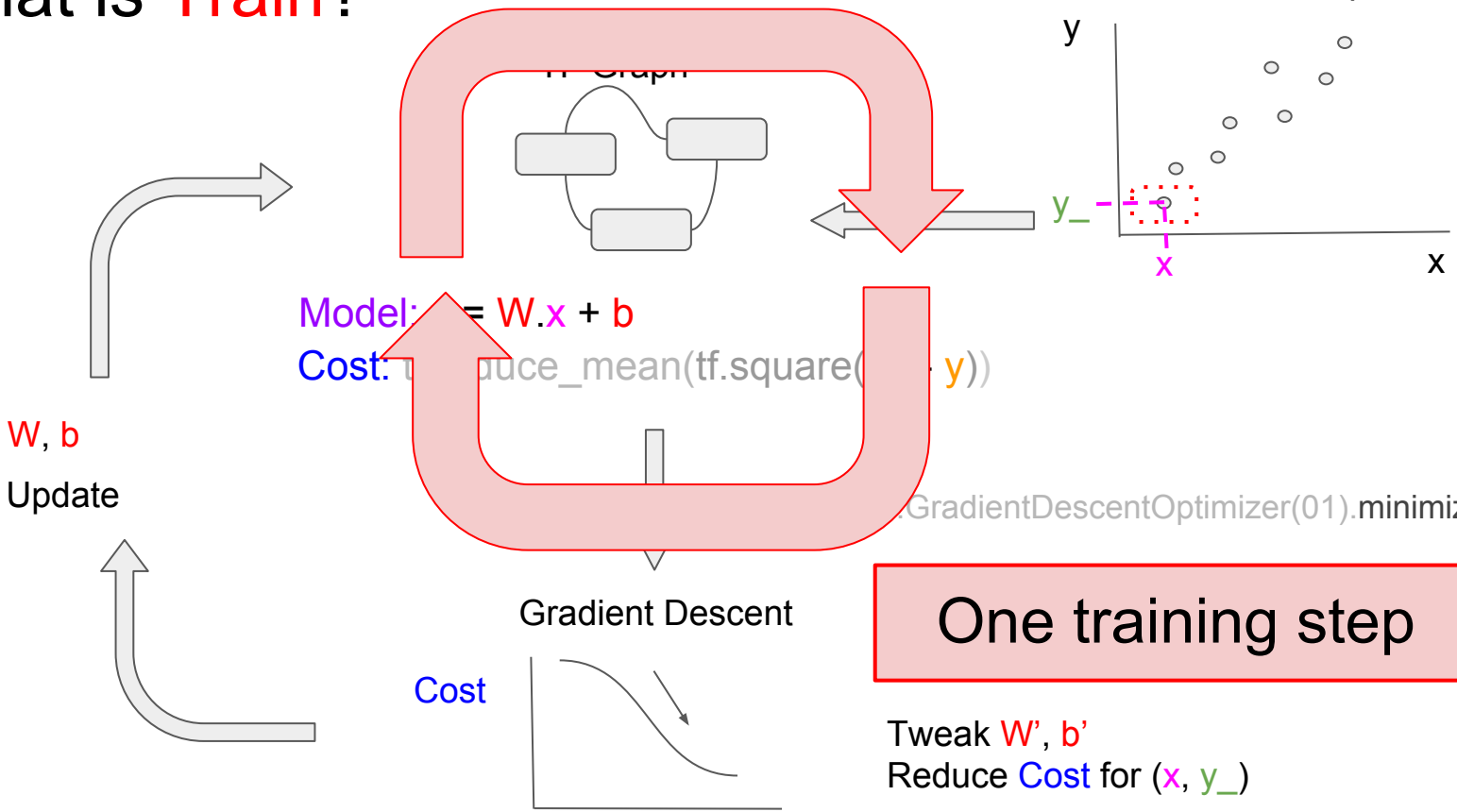
Tweak W', b'

Reduce Cost for (x, y_)

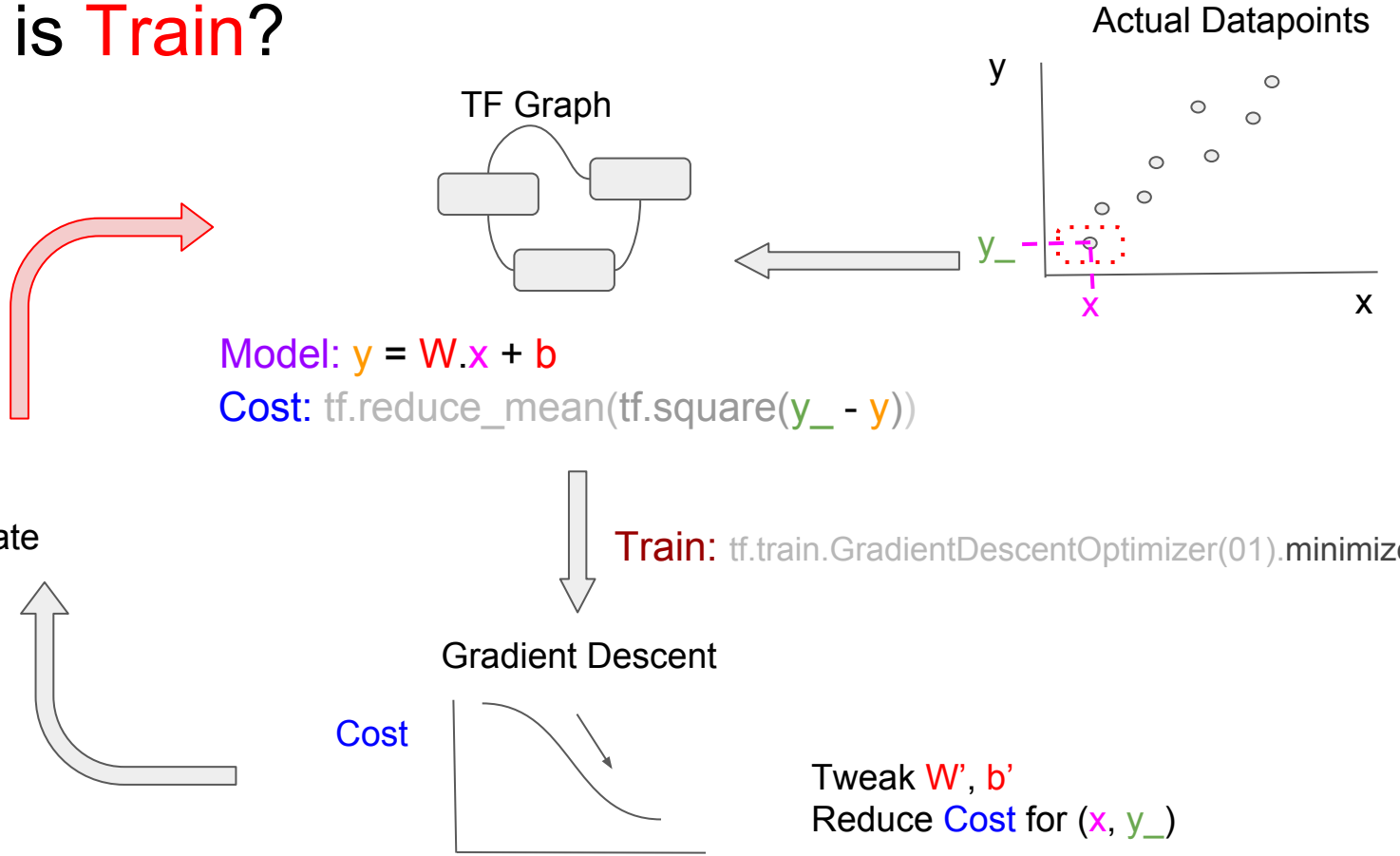
What is Train?



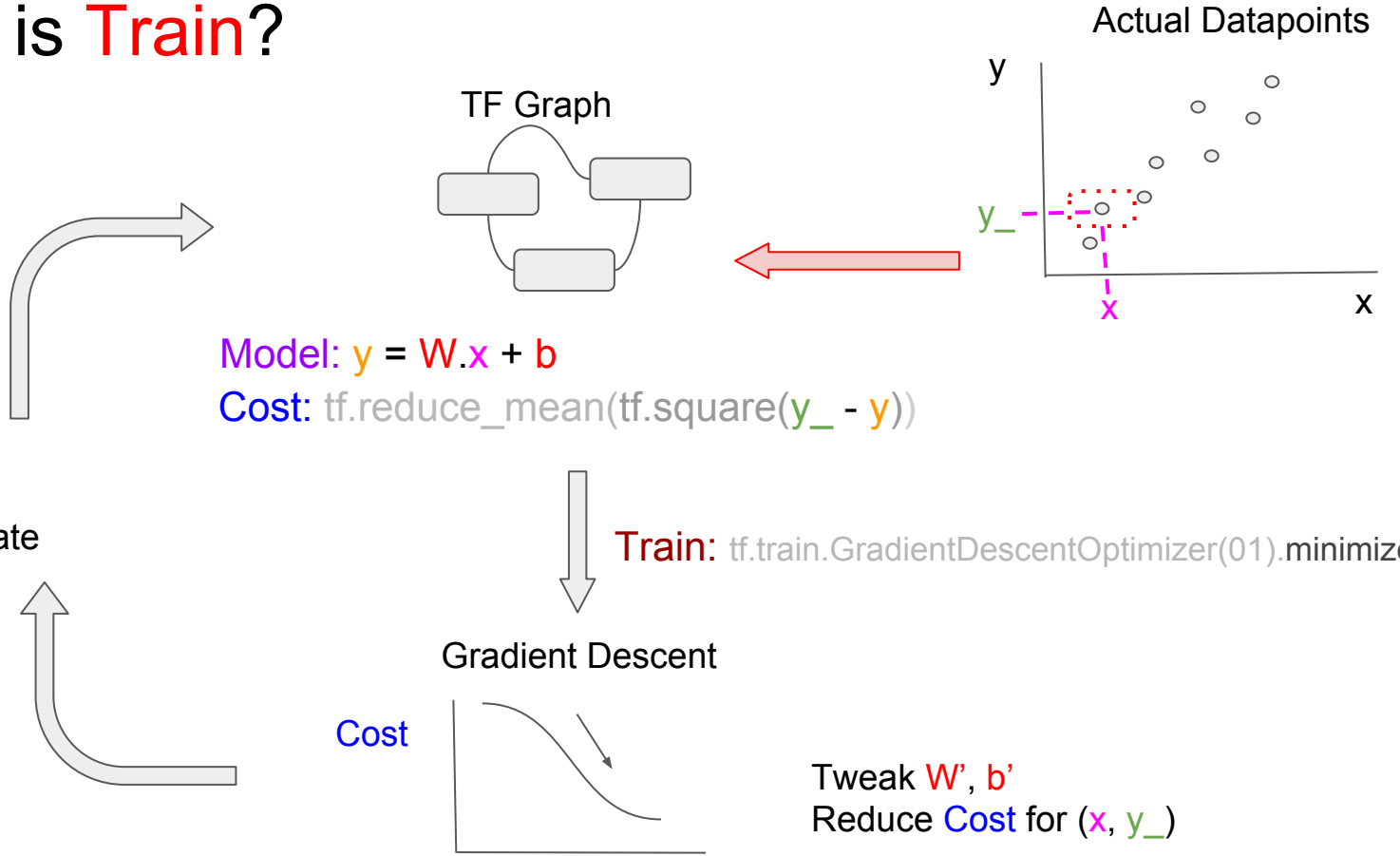
What is Train?



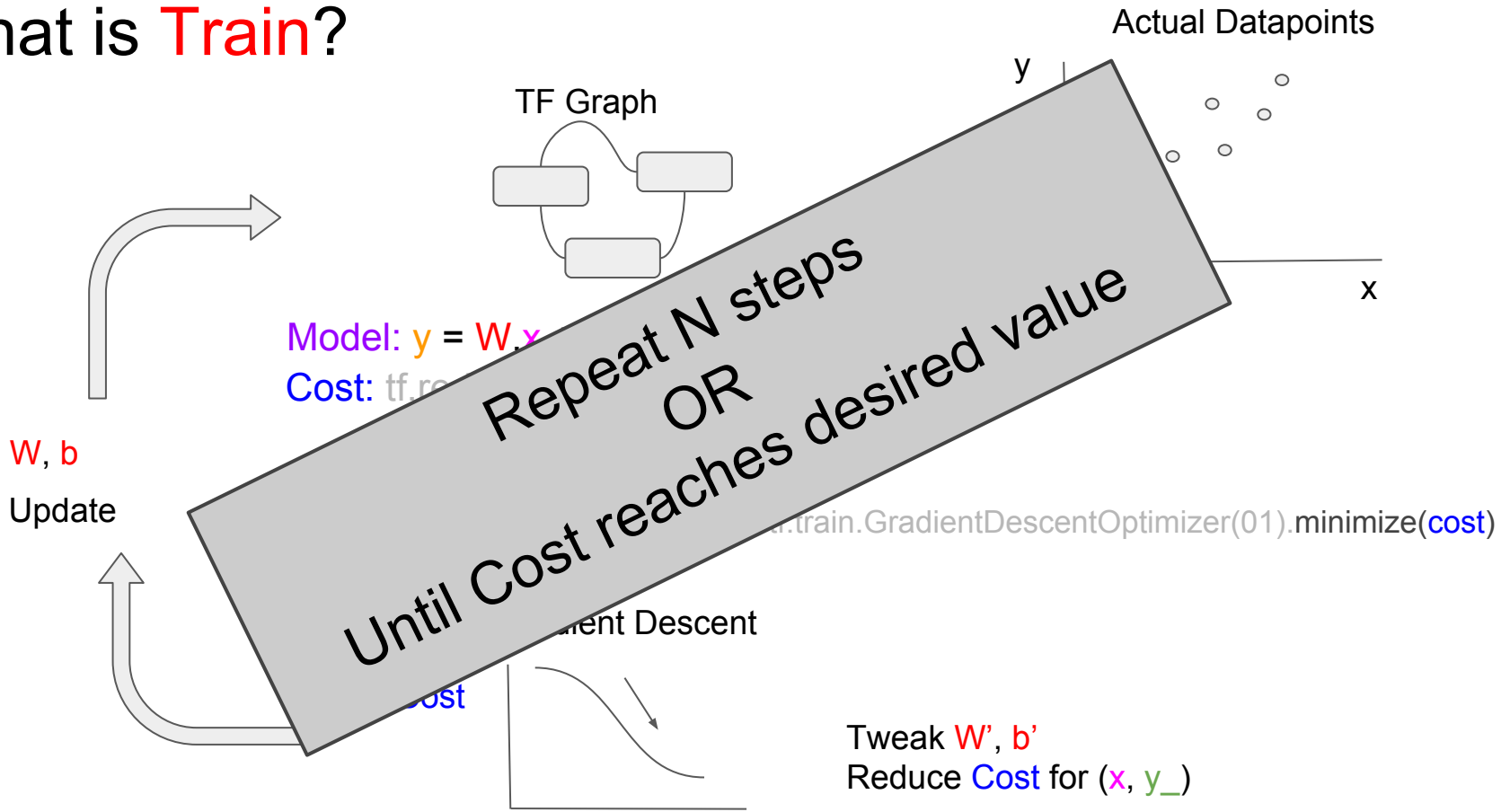
What is Train?



What is Train?

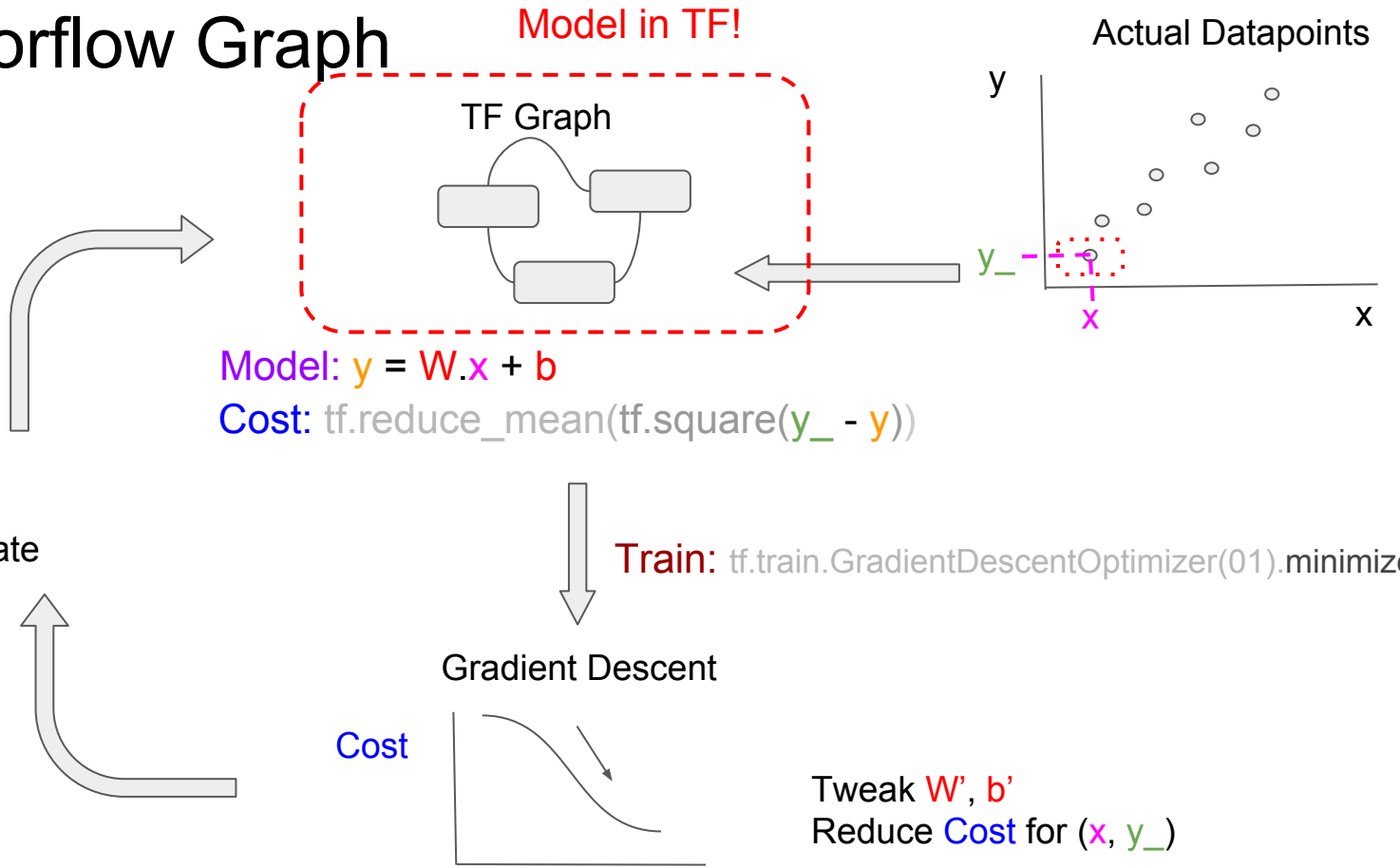


What is Train?

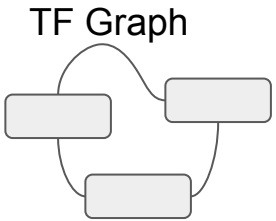


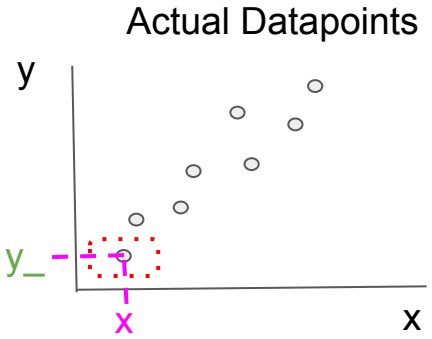
Creating Tensorflow Graph

Tensorflow Graph



Tensorflow Graph





Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

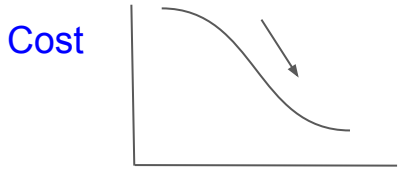
W, b

Update

Define as tf.Variable

Train: tf.train.GradientDescentOptimizer(01).minimize(cost)

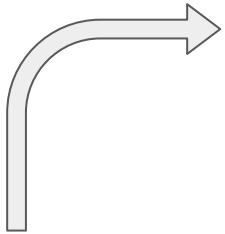
Gradient Descent



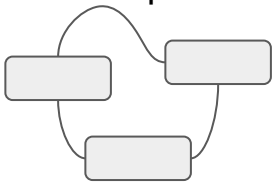
Tweak W', b'

Reduce Cost for (x, y_)

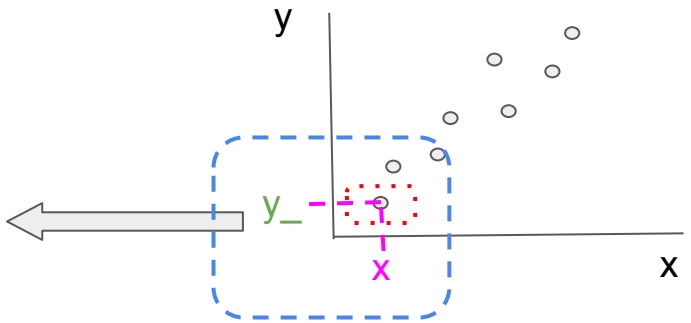
Tensorflow Graph



TF Graph



Actual Datapoints



Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

Define as tf.placeholder

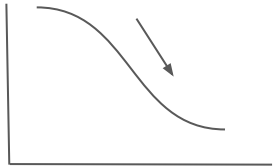
W, b

Update

Train: tf.train.GradientDescentOptimizer(01).minimize(cost)

Gradient Descent

Cost

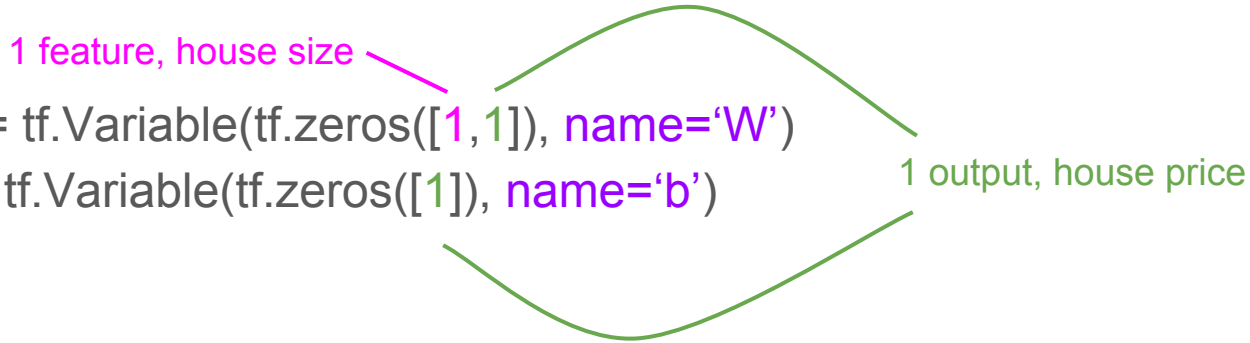


Tweak W', b'

Reduce Cost for (x, y_)

Tensorflow Graph: tf.Variable (Enhance visualization)

tf.Variable for holding variables we want to learn, W, b



NOTE: Without *name=XXX*, the variables will be assigned names like Variable_0, Variable_1, etc.

Tensorflow Graph: tf.placeholder (Enhance visualization)

tf.placeholder for actual data, e.g., house size (x), house price (y_)

1 feature, house size

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

1 output, house price

NOTE: Without *name=XXX*, the placeholders will be assigned names like Placeholder_0, Placeholder_1, etc.

Tensorflow Graph: Model (Enhance visualization)

```
# Model linear regression y = Wx + b

with tf.name_scope("Wx_b") as scope:
  product = tf.matmul(x,W)
  y = product + b
```

NOTE: With *scope*, the all definitions under it, will appear as a black-box that is expandable!

Tensorflow Graph: Cost (Enhance visualization)

```
# Cost function: Least-squared difference

with tf.name_scope("cost") as scope:
  cost = tf.reduce_mean(tf.square(y_-y))
```

Tensorflow Graph: Train (Enhance visualization)

```
# Train step

with tf.name_scope("train") as scope:
  train_step = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

Tensorboard

Two Main Components

- Summary
  - Mark data we want to collect
    - W, b, cost
- Writer
  - Write data collected to event file

Tensorflow Summary: Scalar

tf.scalar_summary

```
# cost is single-dimension so
# mark it as data to be collected using scalar_summary

cost = tf.reduce_mean(tf.square(y_-y))
cost_sum = tf.scalar_summary("cost", cost)
```

Tensorflow Summary: Histogram

tf.histogram_summary

```
# W and b can be multi-dimensional so
# mark them as data to be collected using histogram_summary

W = tf.Variable(tf.zeros([1,1]), name="W")
W_hist = tf.histogram_summary("weights", W)

b = tf.Variable(tf.zeros([1]), name="b")
b_hist = tf.histogram_summary("biases", b)
```

Tensorflow Summary: Merge

tf.merge_all_summaries

merged = tf.merge_all_summaries()

tf.merge_summary

merged = tf.merge_summary([W_hist, b_hist, cost_sum])

Tensorflow Writer

```
# The specified log directory will be:
# * Created if it does not exist
# * If exist and not empty data is merged
# sess.graph_def will enable Tensorflow to draw graph representation
# for network

writer = tf.train.SummaryWriter("/event_log_dir", sess.graph_def)
```

Tensorflow Writer: When to Record Data

```
# NOTE: Train using #steps
for i in steps:
  ...
  # NOTE: Write summary every 10 samples
  if i % 10 == 0:
    all_feed = { x: all_xs, y_: all_ys }
    # NOTE:
    # * merged is our merged summary name
    # * feed is the batch of all  x, y data points, to calculate TOTAL cost
    result = sess.run(merged, feed_dict=all_feed)
    writer.add_summary(result, i)
  ...
```

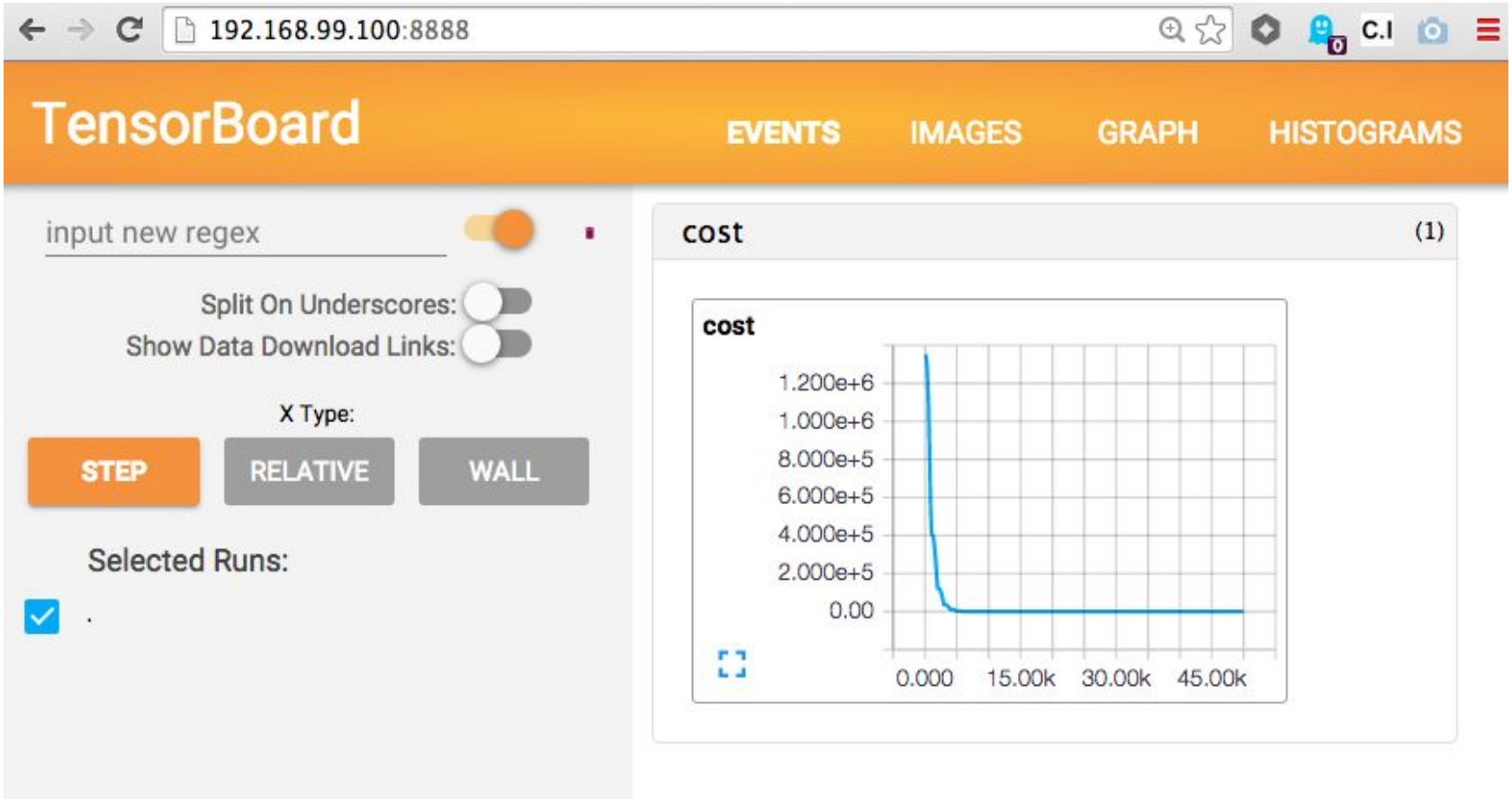
Start Tensorboard Server

NOTE: Single line

python /usr/local/lib/python2.7/dist-packages/tensorflow/tensorboard/tensorboard.py --logdir /event_log_dir/ --port 8888

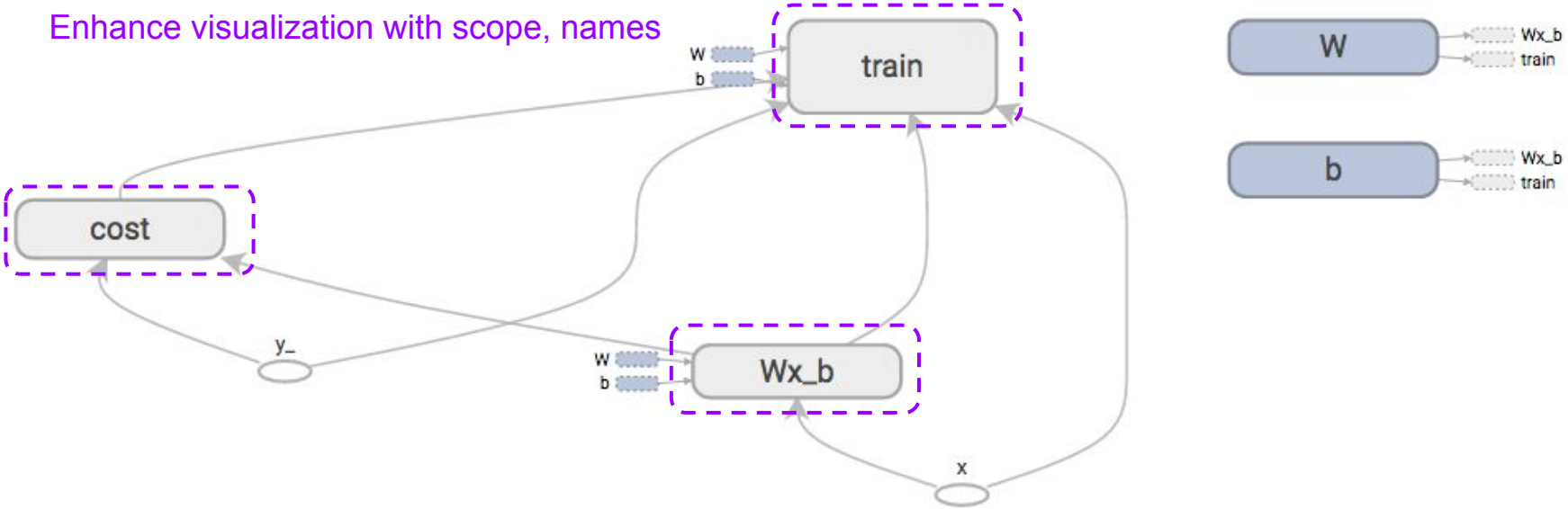
WARNING:tensorflow:Found more than one graph event per run.Overwritting the graph with the newest event

Access Using Browser

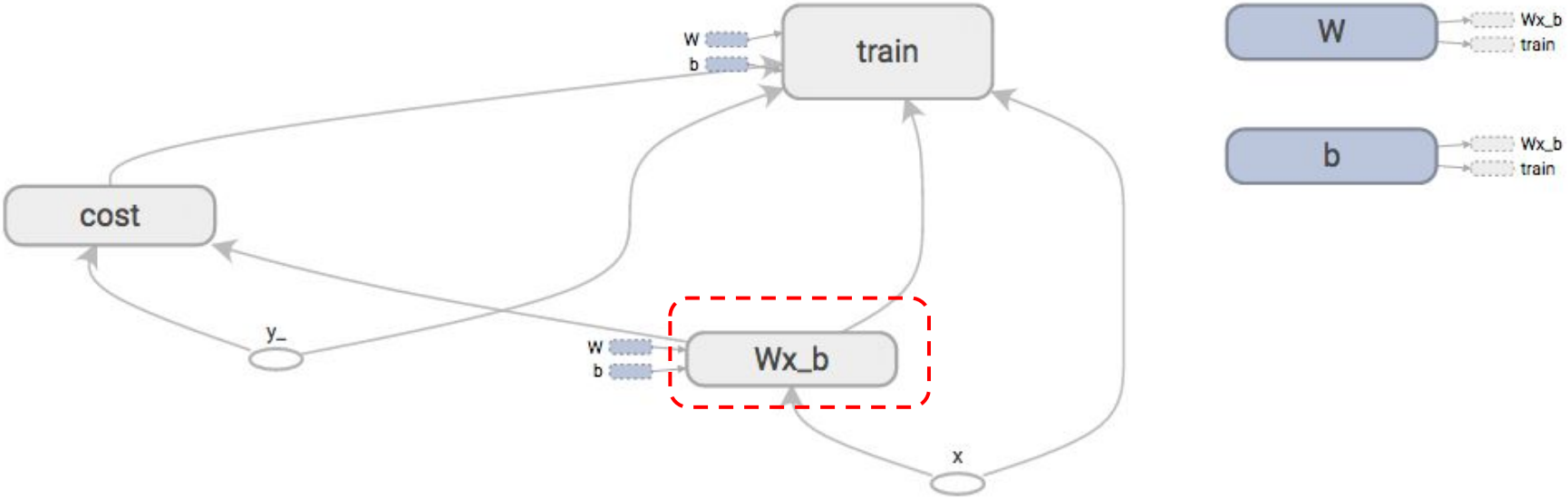


Tensorflow Graph

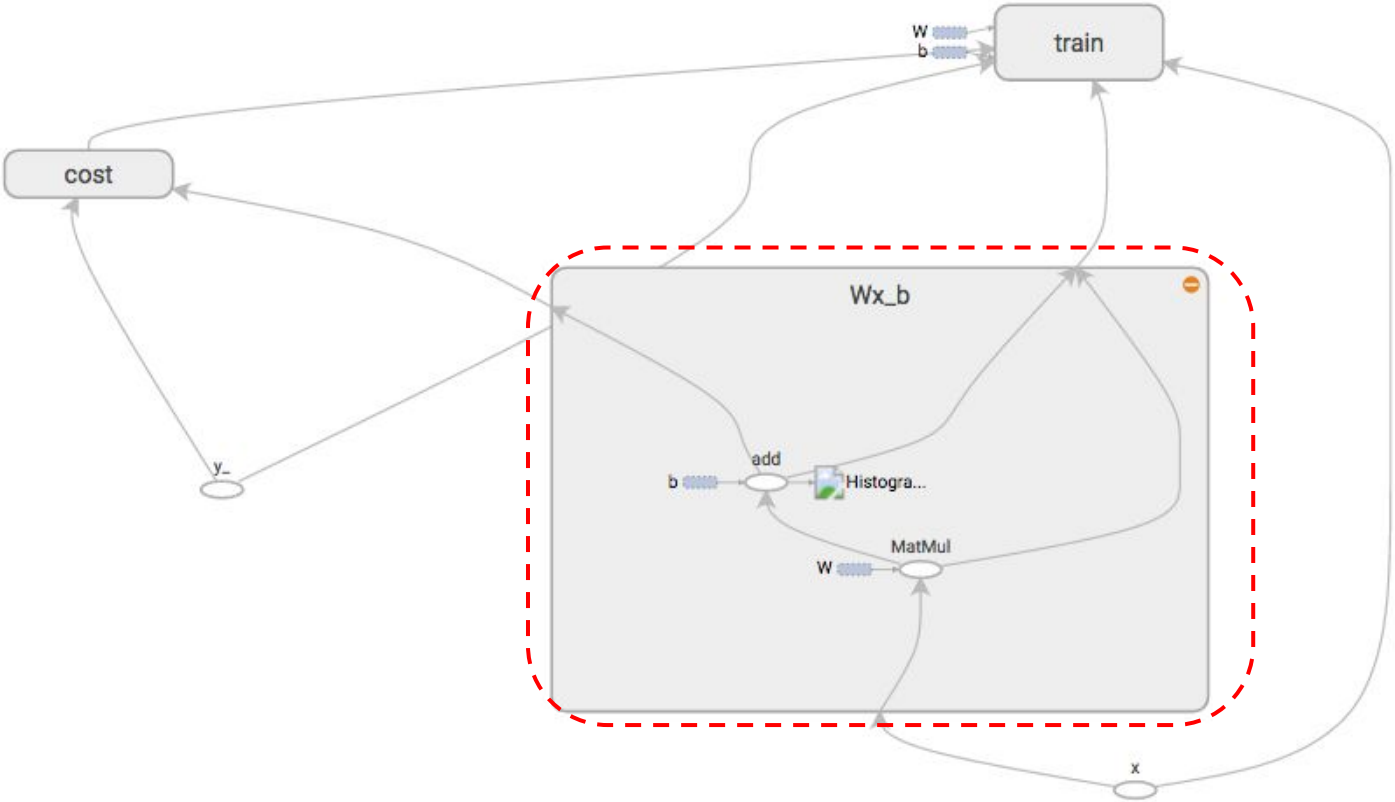
Enhance visualization with scope, names



Tensorflow Graph

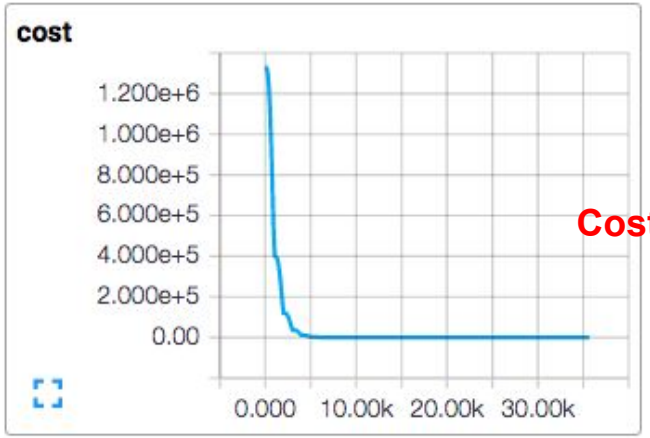


Tensorflow Graph: Expanded



Importance of Visualization: Have we found a good fit?

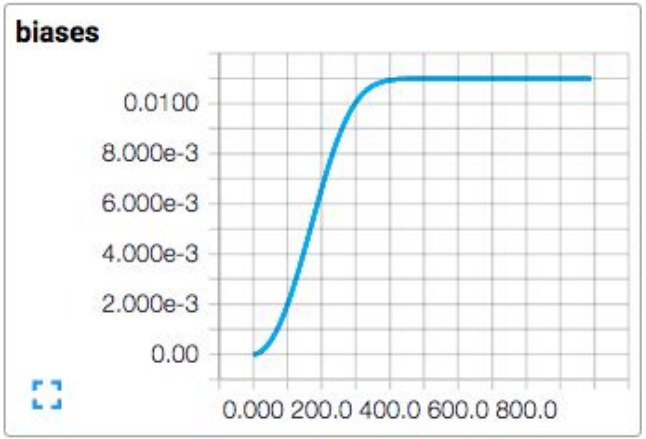
Linear Regression with 1-Feature

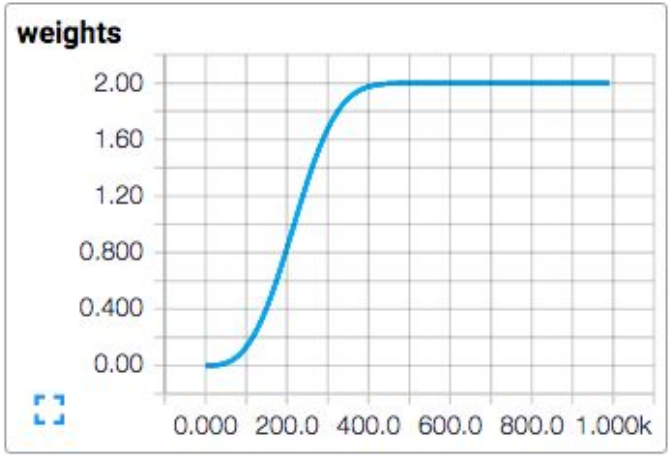


Cost reaches zero = good model fit

b ~ 0.0

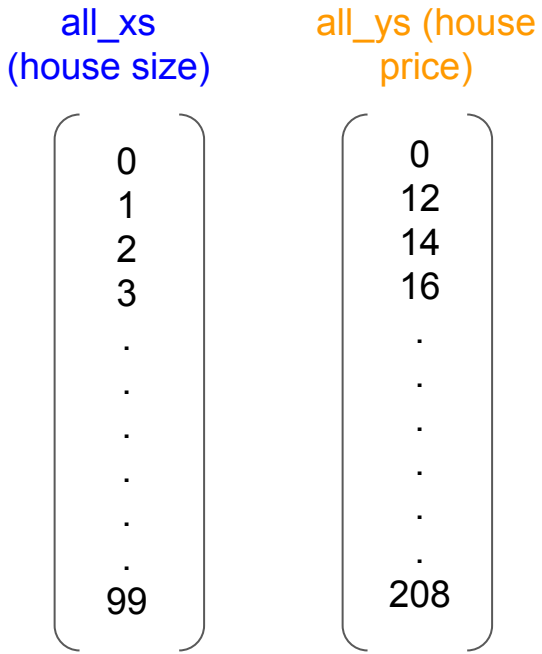
W ~ 2.0



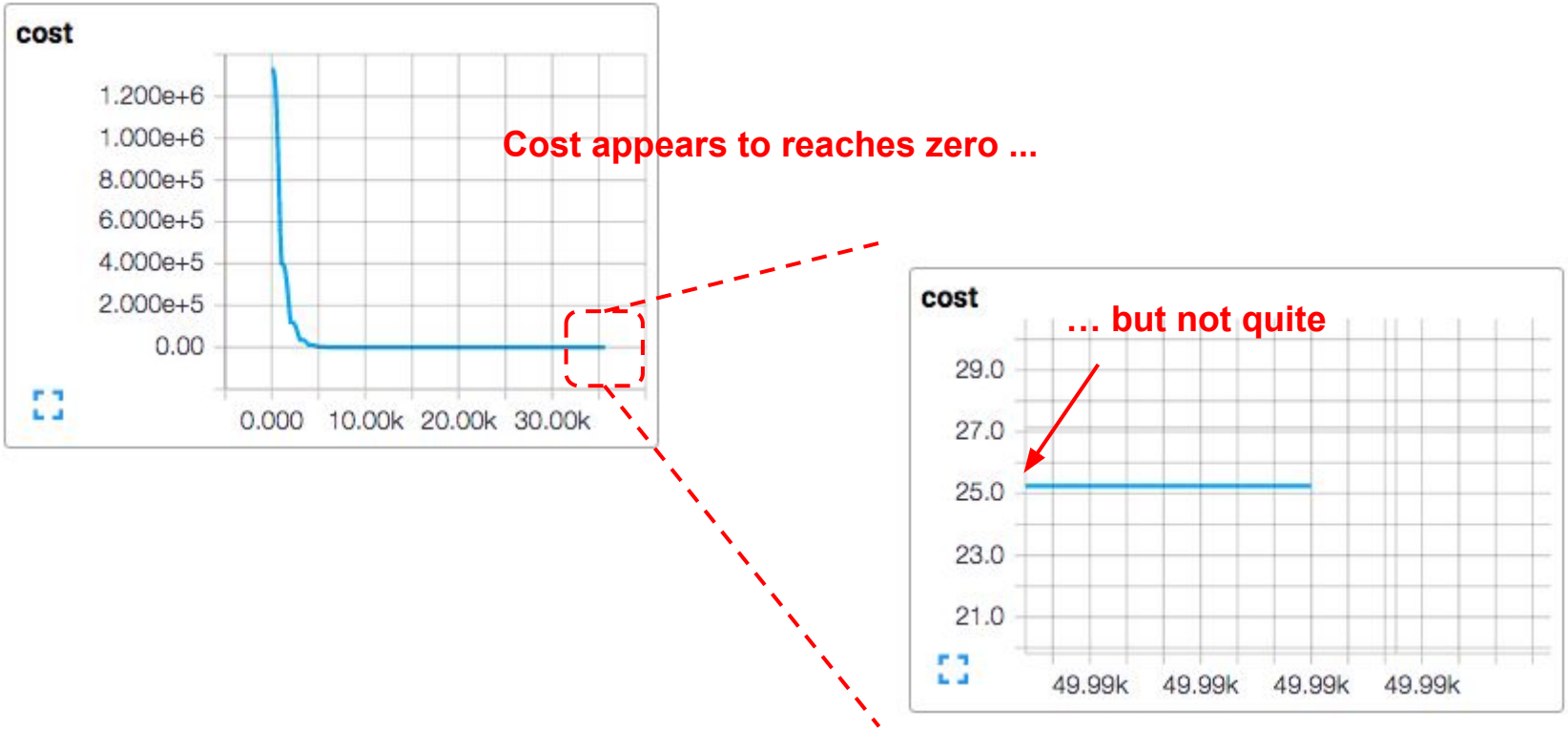


Training Linear Regression with b > 0

```
# Generate fake data for y = 2.x + 10, i.e., W = 2, b = 10

all_xs = []
all_ys = []
for i in range(datapoint_size):
  # Create fake data for y = W.x + b where W = 2, b = 10
  all_xs.append(i)
  all_ys.append(2*i+10)
```



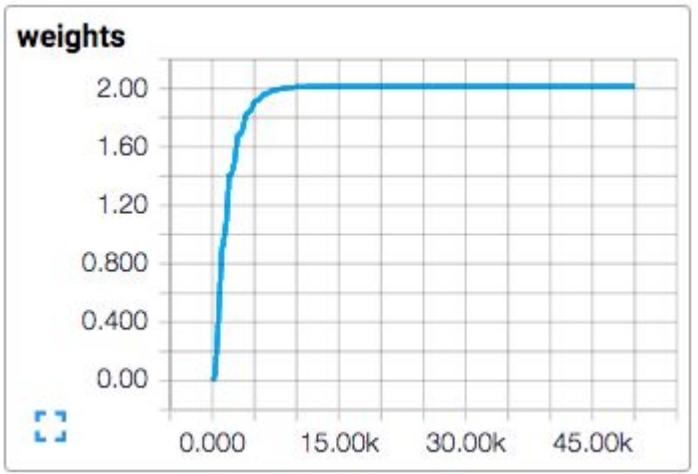
Linear Regression with 1-Feature (b > 0)



Linear Regression with 1-Feature (b > 0)

W ~ 2.0

b slowly learning … (ideally 10)

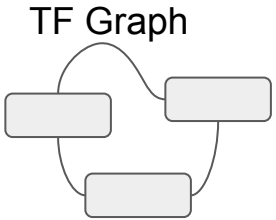


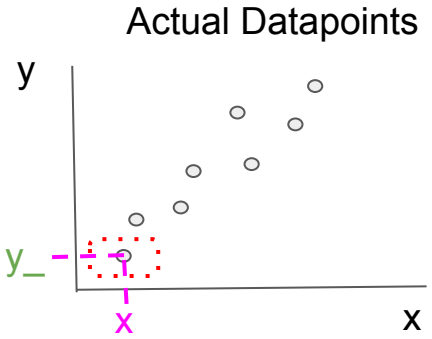
Is there a way to learn faster?

- Faster gradient descent learning rate?
- Different model?
- Different cost function?

Batch Size

Stochastic Gradient Descent



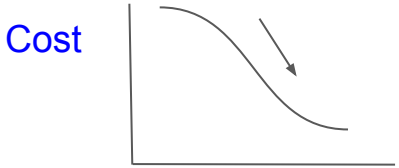


Model: y = W.x + b

Cost: tf.reduce_mean(tf.square(y_ - y))

Train: tf.train.GradientDescentOptimizer(01).minimize(cost)

Gradient Descent

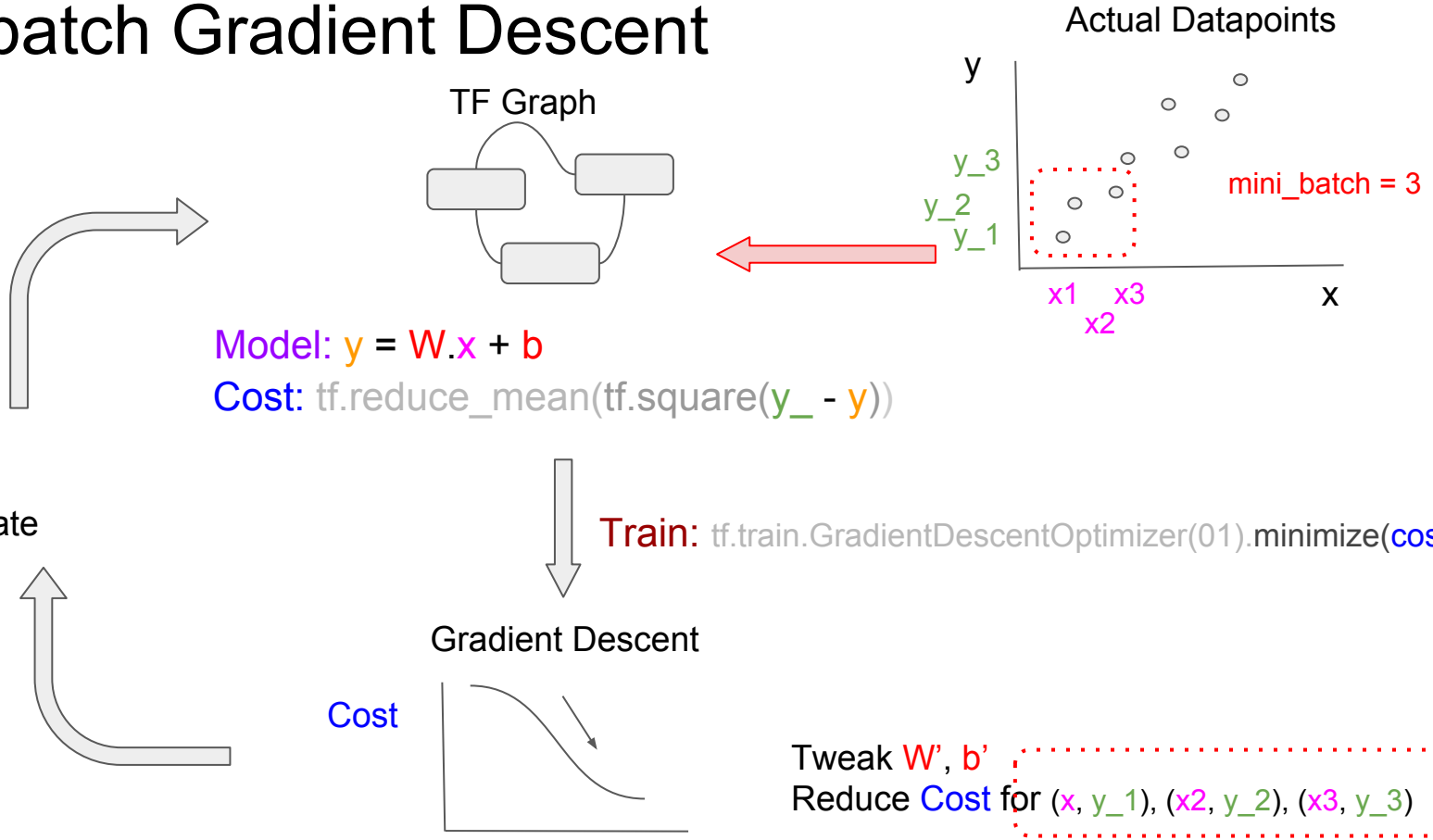


Tweak W', b'
Reduce Cost for (x, y_)

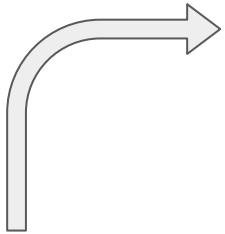
W, b

Update

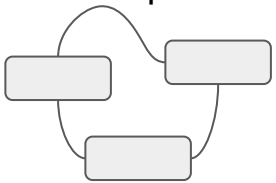
Mini-batch Gradient Descent

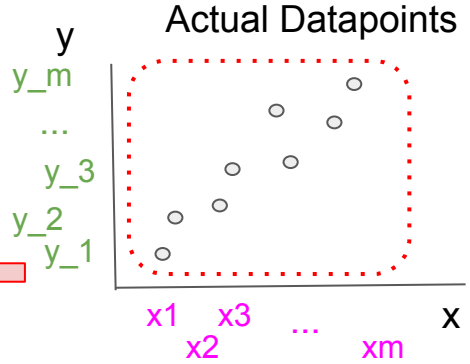


Batch Gradient Descent



TF Graph





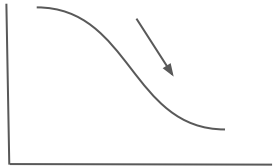
Model: y = W.x + b

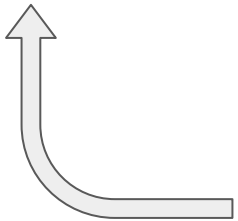
Cost: tf.reduce_mean(tf.square(y_ - y))

Train: tf.train.GradientDescentOptimizer(01).minimize(cost)

Gradient Descent

Cost





W, b

Update

Tweak W', b'

Reduce Cost for (x, y_1), (x2, y_2), ... (xm, y_m)

Stochastic/Mini-batch/Batch Gradient Descent

One feature, house size

```
x = tf.placeholder(tf.float32, [None, 1])
y_ = tf.placeholder(tf.float32, [None, 1])
...
train_step = tf.train.GradientDescentOptimizer(01).minimize(cost)
```

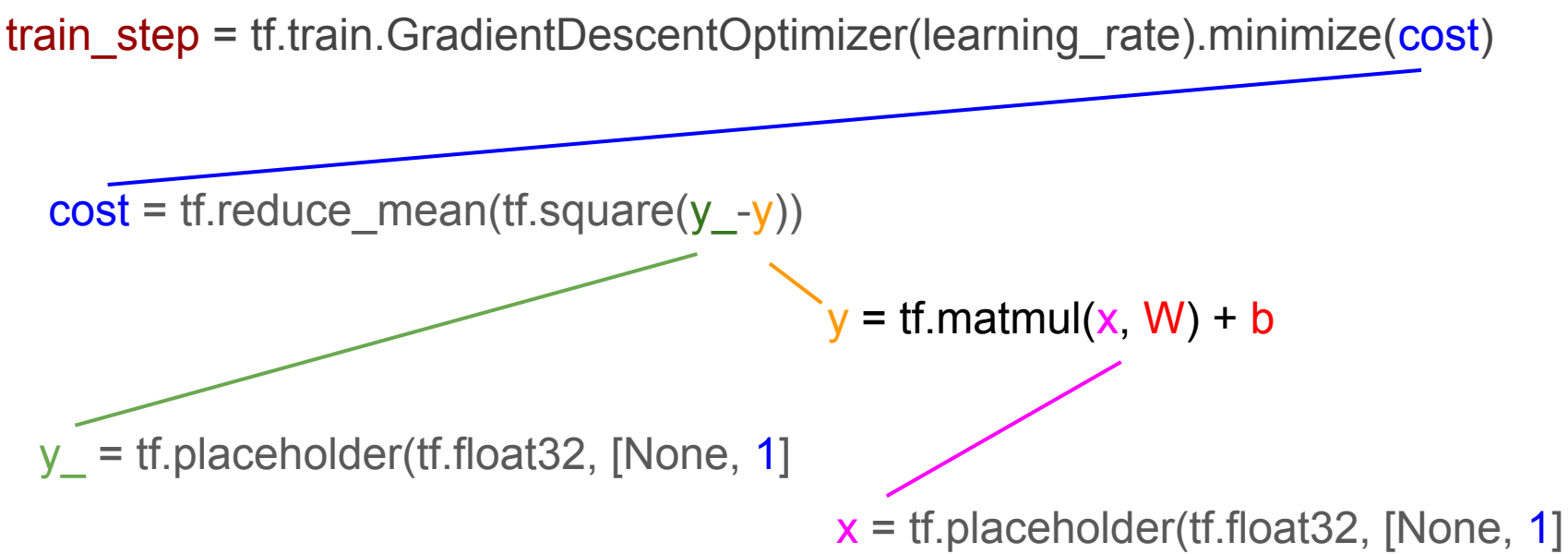
One output, house price

```
xs = ....
ys = ....
```

ONLY this part is different!

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

Why Feed train_step, with feed_dict = x, y_



train_step has dependency on x, y_

Stochastic Gradient Descent

Stochastic Gradient Descent

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

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]])

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

One feature, house size

One output, house price

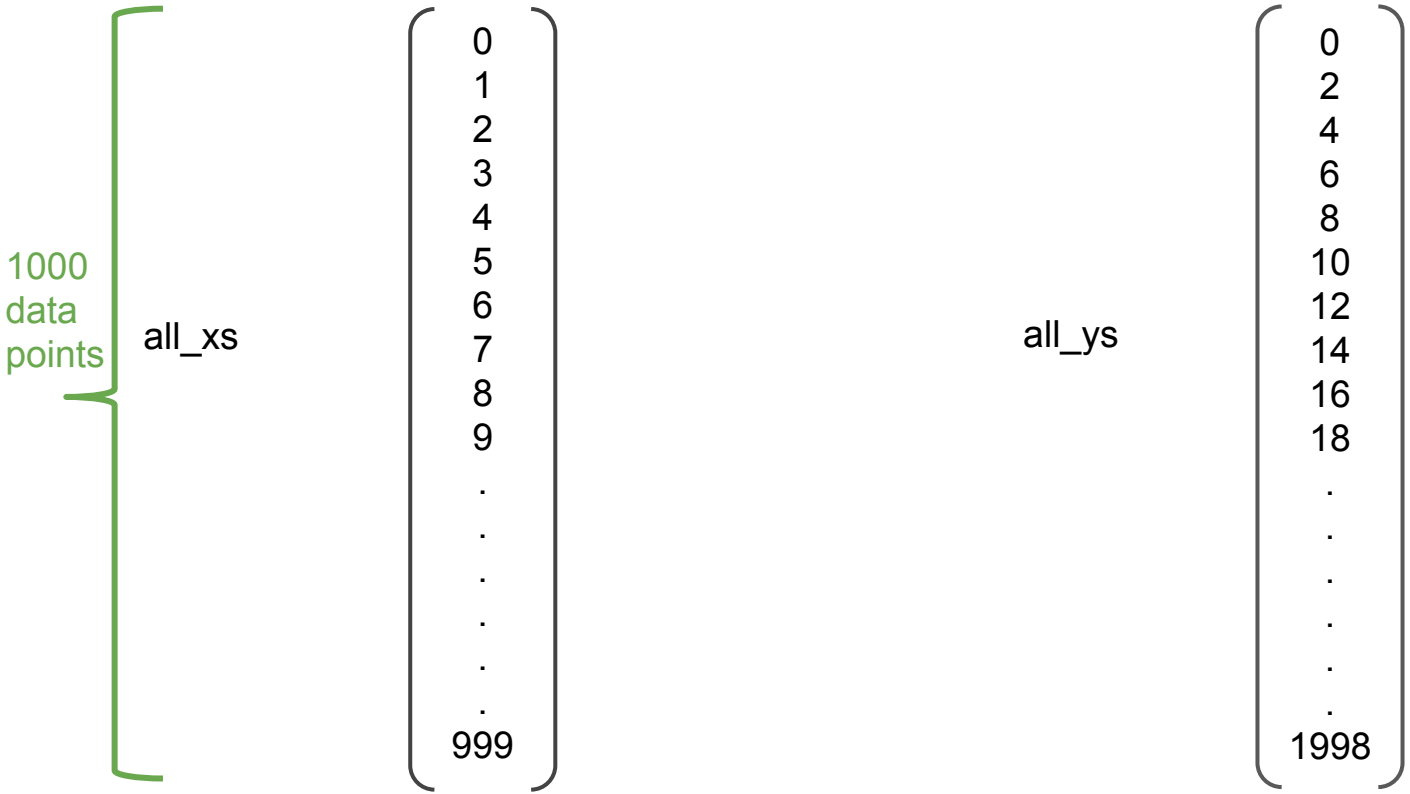
Provide datapoint for feature, house size each step

Provide datapoint for output, house price, each step

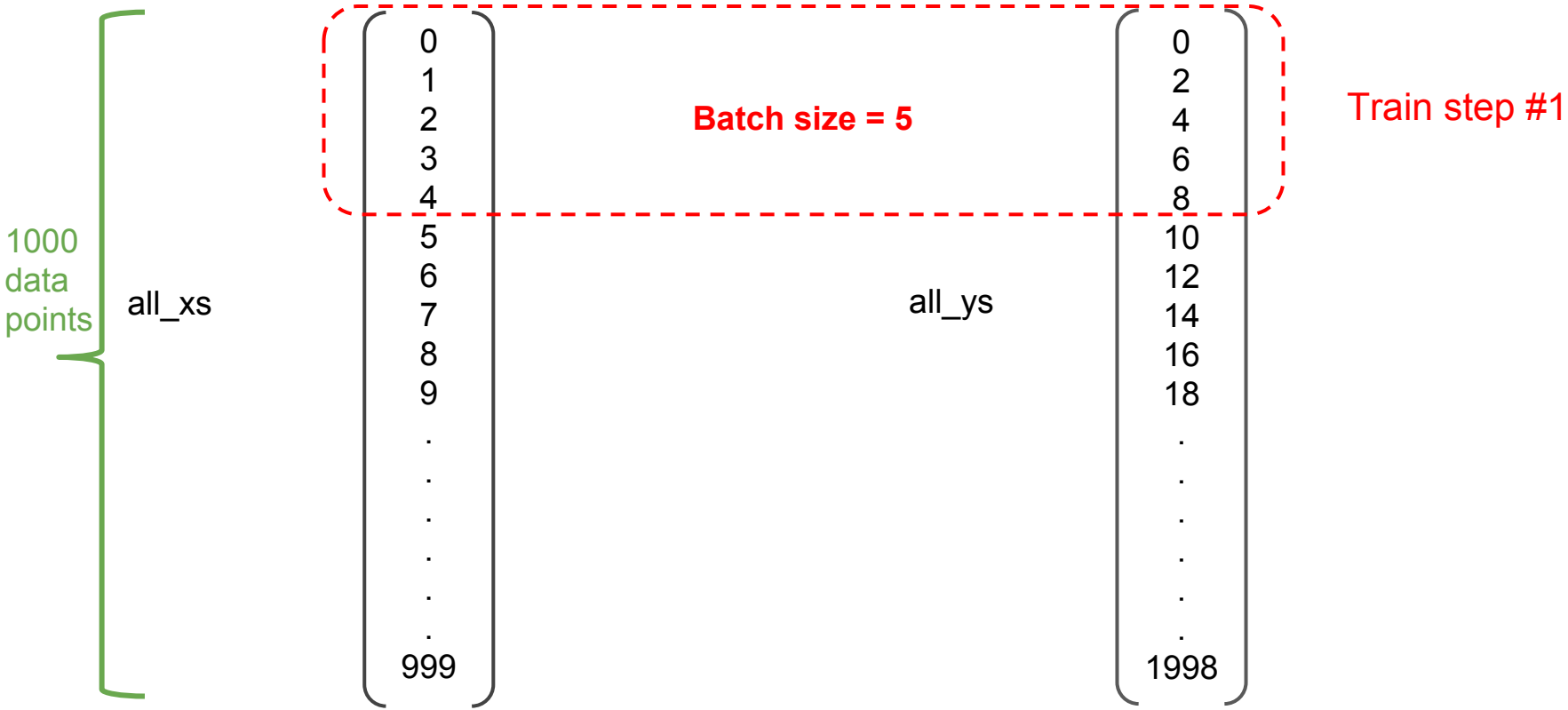
Stochastic Gradient Descent

Mini-batch Gradient Descent

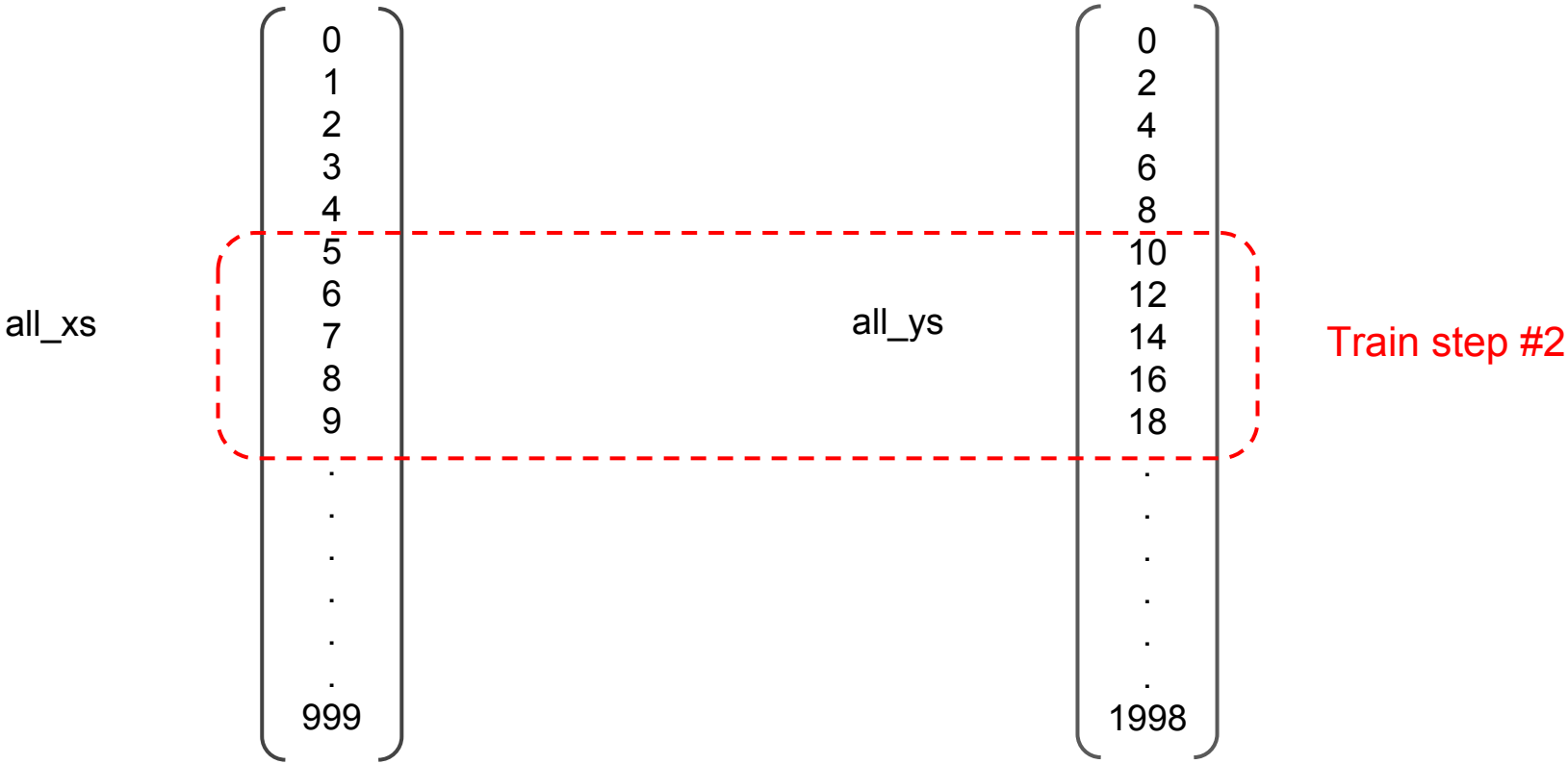
Mini-batch Gradient Descent: x, y_



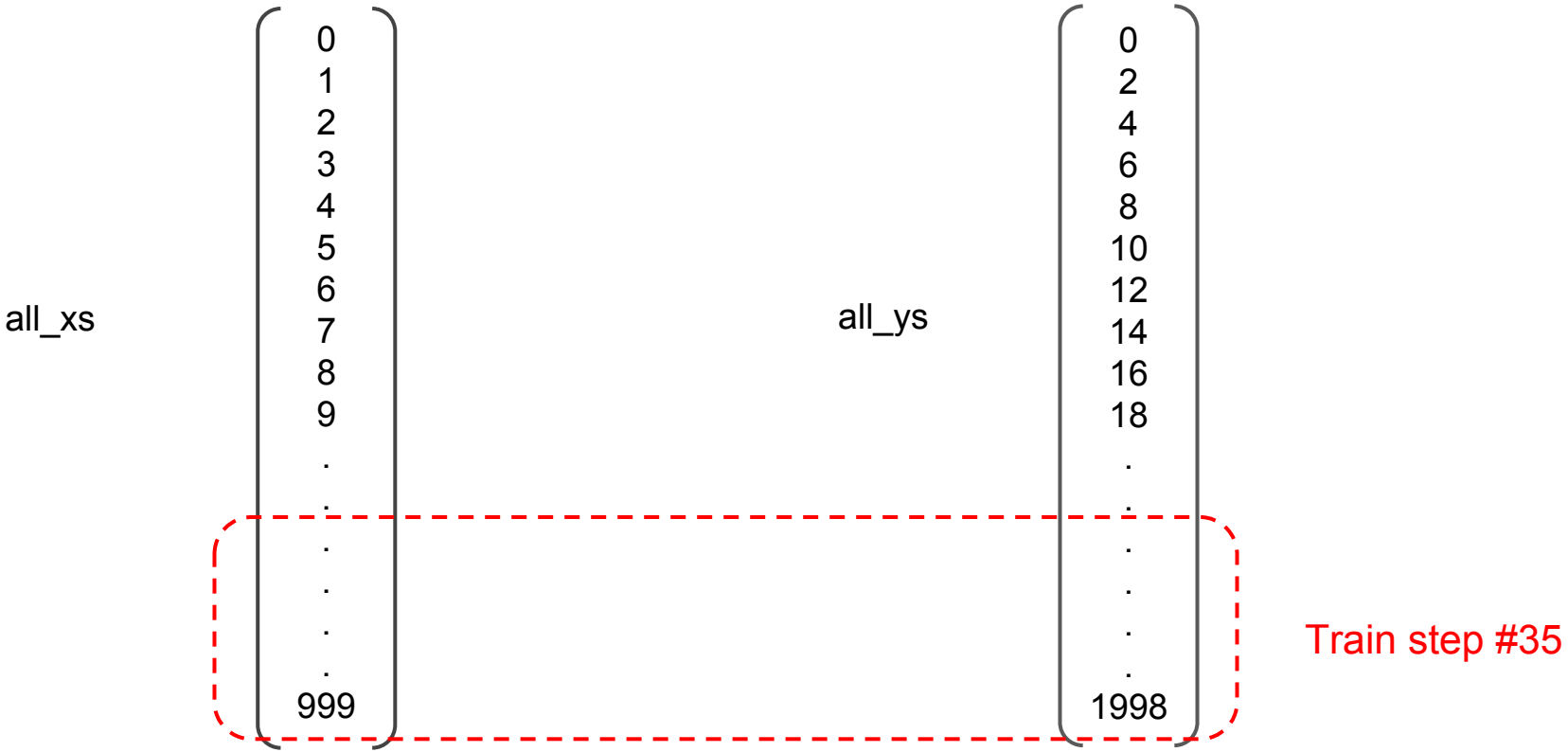
Mini-batch Gradient Descent: x, y_



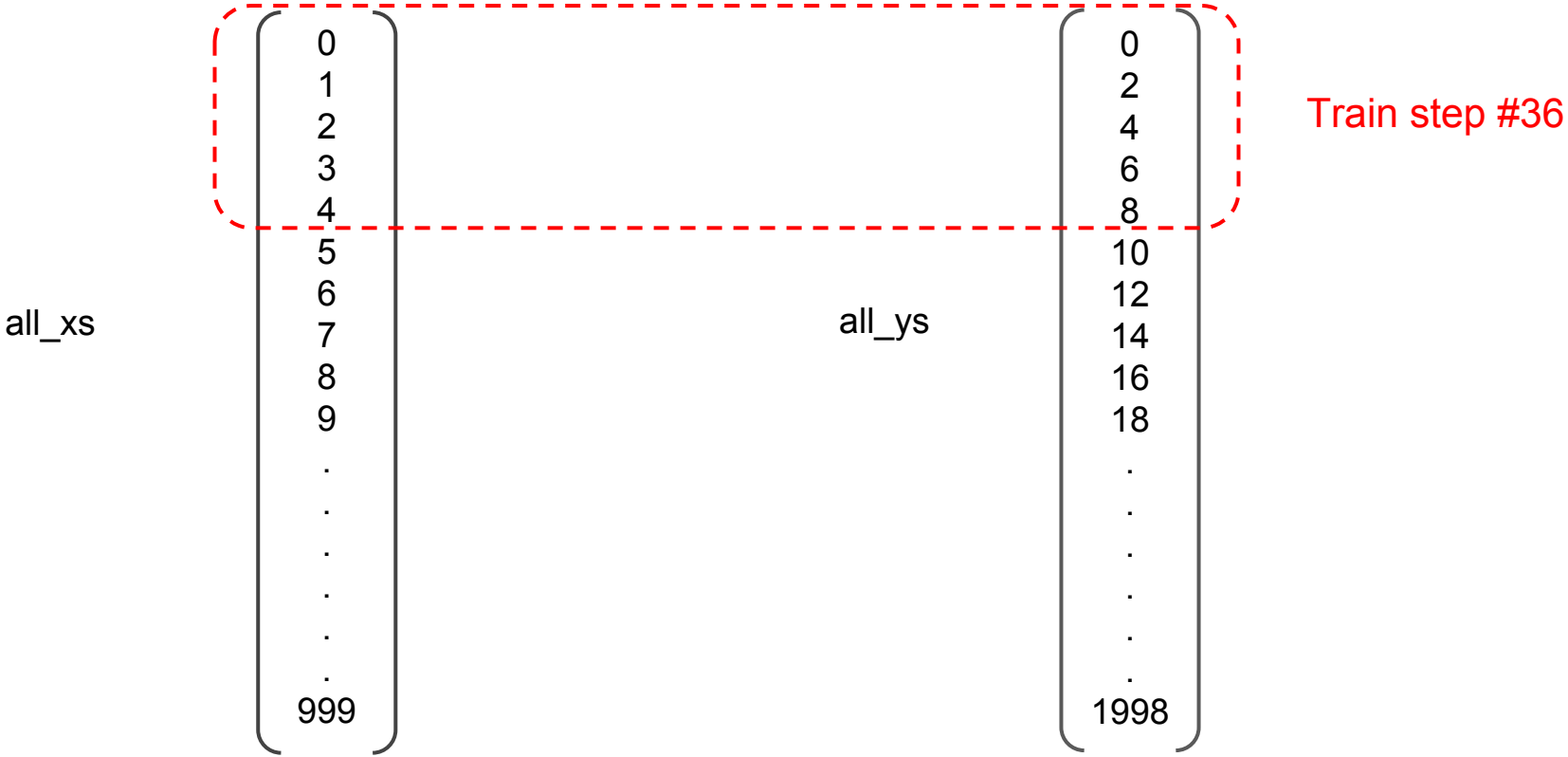
Mini-batch Gradient Descent: x, y_



Mini-batch Gradient Descent: x, y_



Mini-batch Gradient Descent: x, y_



Mini-batch Gradient Descent

One feature, house size

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

One output, house price

```
# Prepare all actual house size (x), house price (y_) before hand
all_xs = ...
all_ys = …

for i in range(steps):
  xs = np.array(all_xs[batch_start_idx:batch_end_idx])
  ys = np.array(all_ys[batch_start_idx:batch_end_idx])
  feed = { x: xs, y_: ys }
  sess.run(train_step, feed_dict=feed)
```

Code to do data batching

Mini-batch Gradient Descent

One feature, house size

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

One output, house price

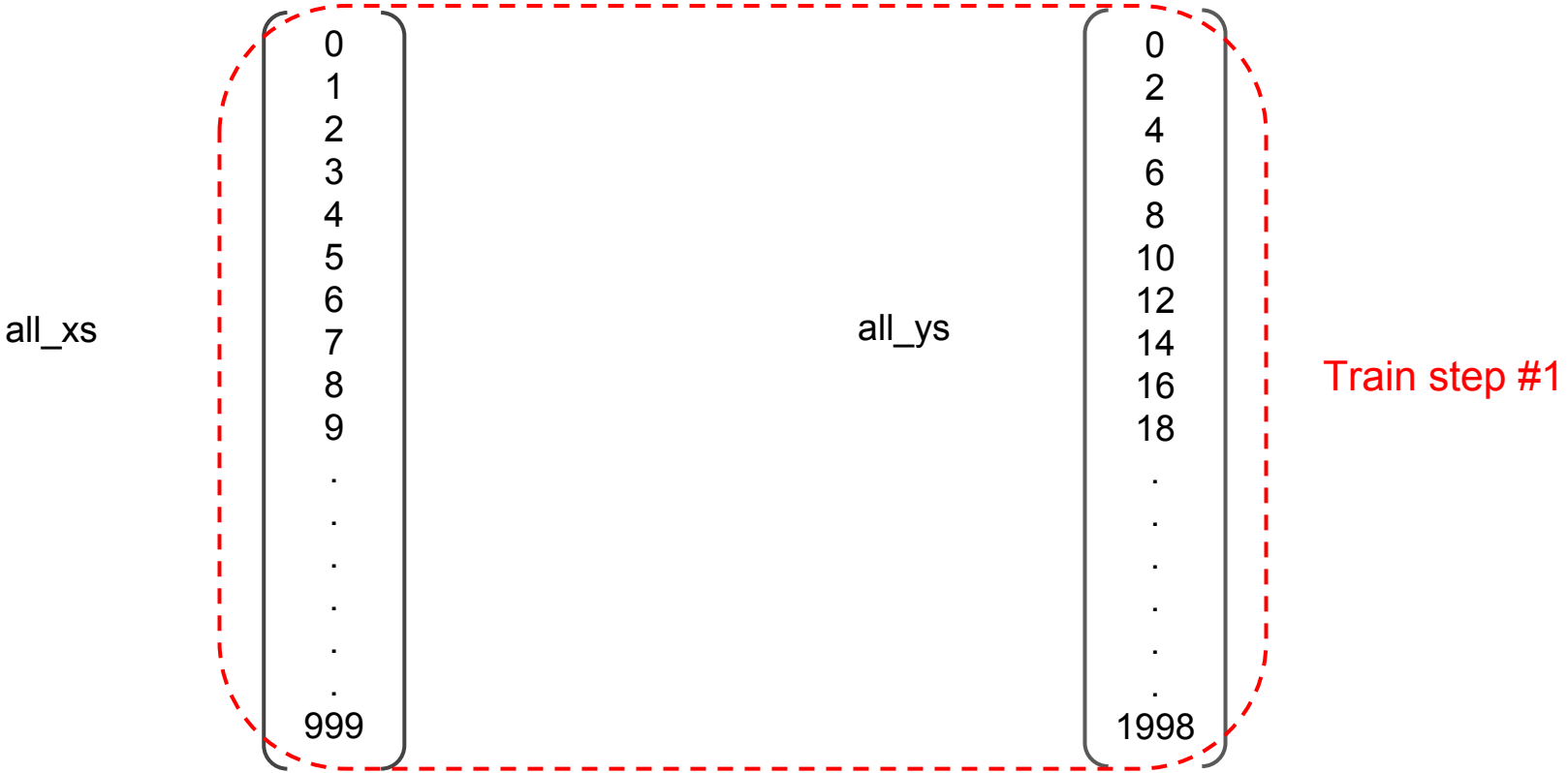
```
# Prepare all actual house size (x), house price (y_) before hand
all_xs = ...
all_ys = …

for i in range(steps):
  xs = np.array(all_xs[batch_start_idx:batch_end_idx])
  ys = np.array(all_ys[batch_start_idx:batch_end_idx])
  feed = { x: xs, y_: ys }
  sess.run(train_step, feed_dict=feed)
```

'None' in placeholder means determined at run-time

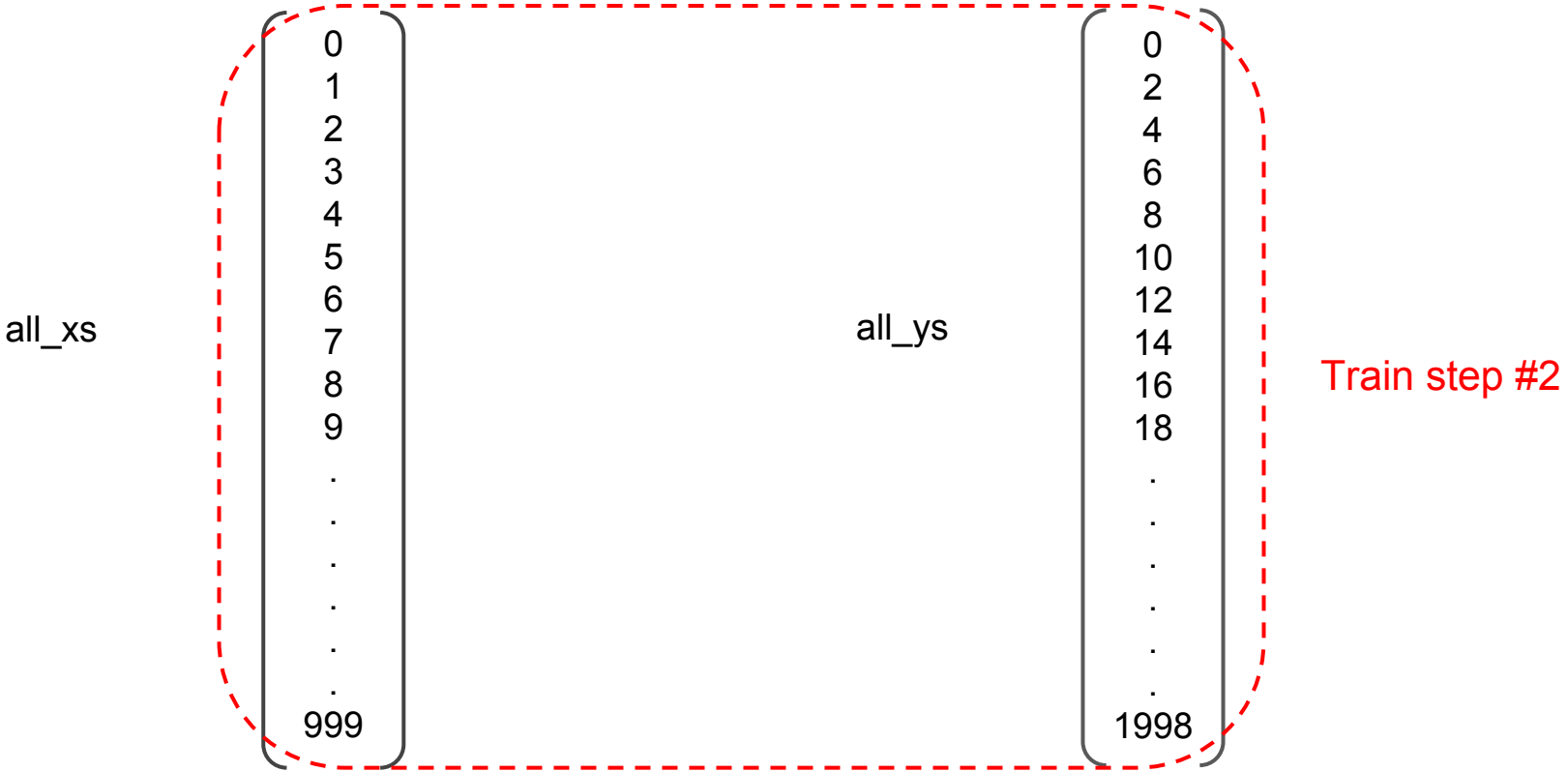
Code to do data batching

Batch Gradient Descent

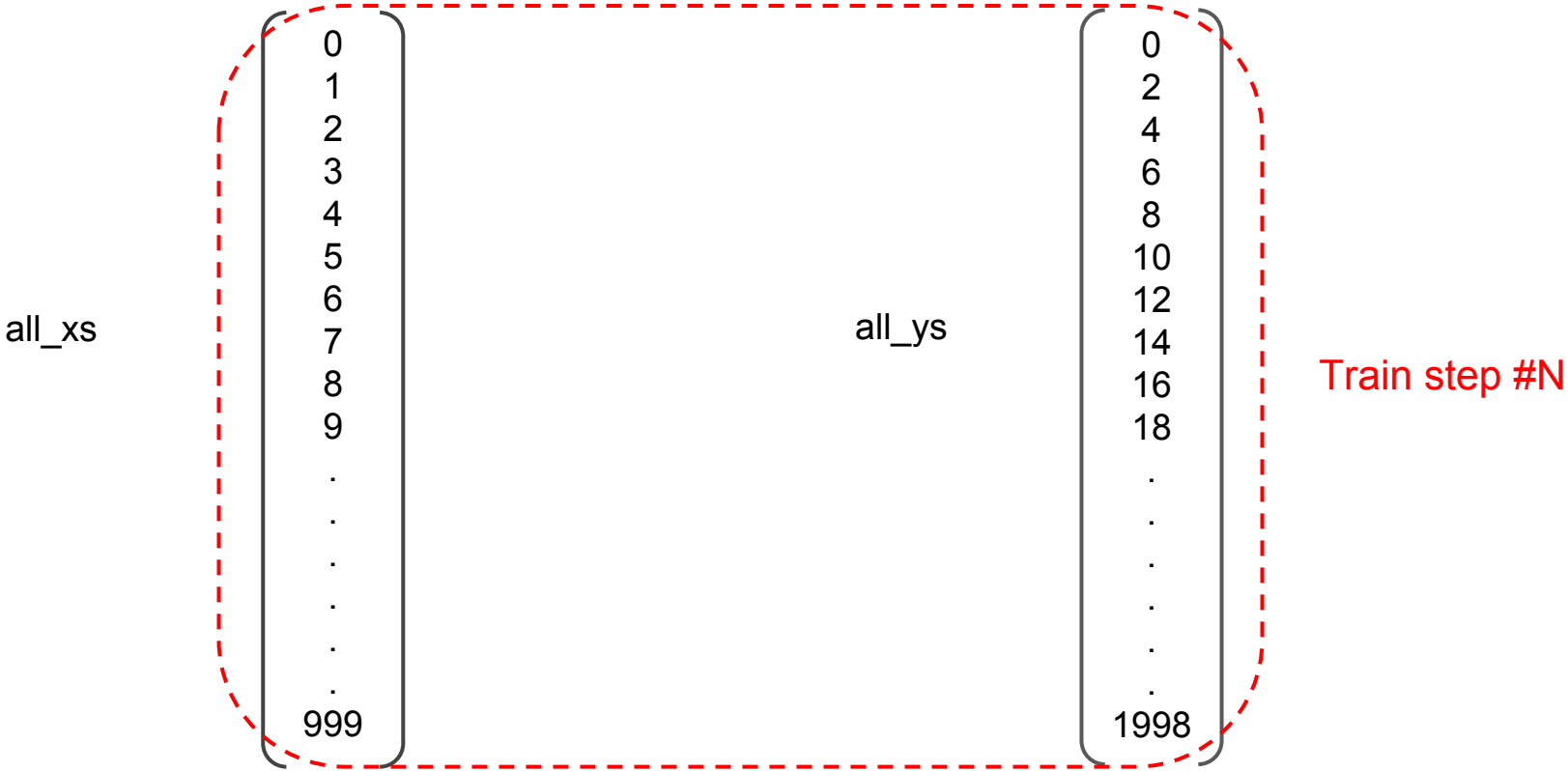
Batch Gradient Descent: x, y_



Batch Gradient Descent: x, y_



Batch Gradient Descent: x, y_



Batch Gradient Descent

One feature, house size

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

One output, house price

```
# Prepare all actual house size (x), house price (y_) before hand
all_xs = ...
all_ys = …

for i in range(steps):
  xs = all_xs
  ys = all_ys
  feed = { x: xs, y_: ys }
  sess.run(train_step, feed_dict=feed)
```

What is the Significance of Batch Size?

Find steepest gradient descent (train_step) from the viewpoint of the points in the batch

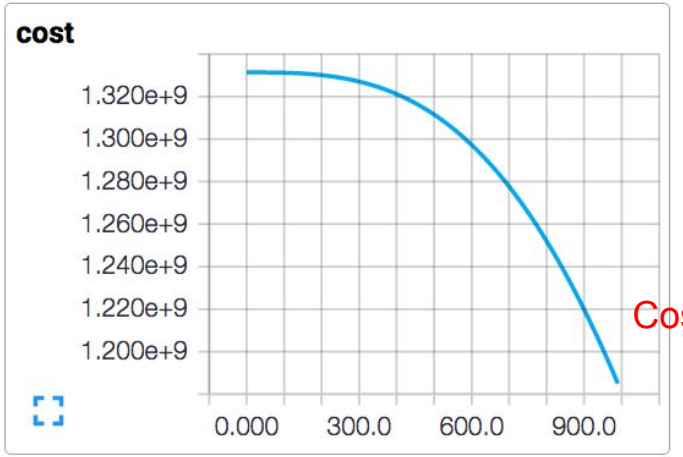
Resource to train each step

Less More

[Batch Size 1] [Batch Size 100] [Full Batch]

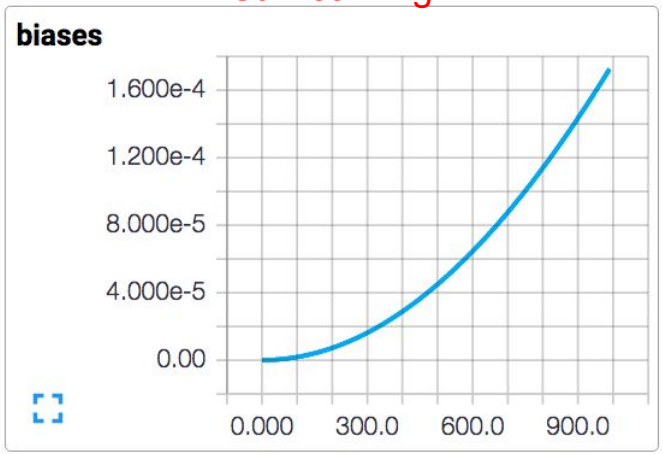
Outcome: With same Gradient Descent learning rate ...

Batch Size = 1, Train Steps = 1000

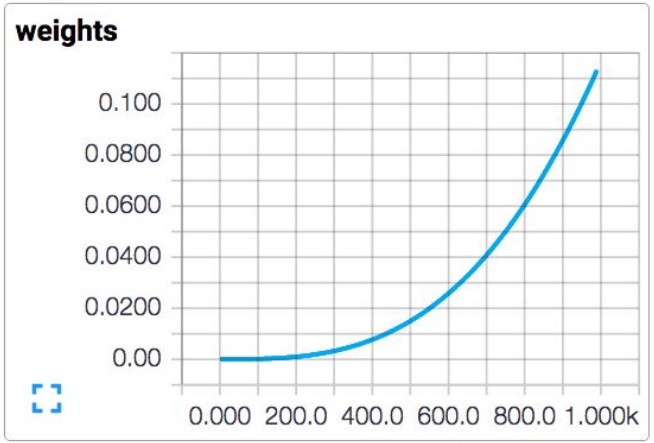


Cost still decreasing towards 0

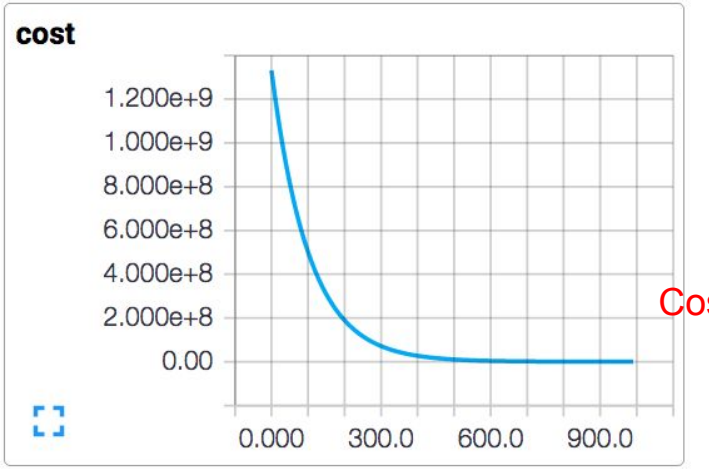
Still learning ...



Still learning ...



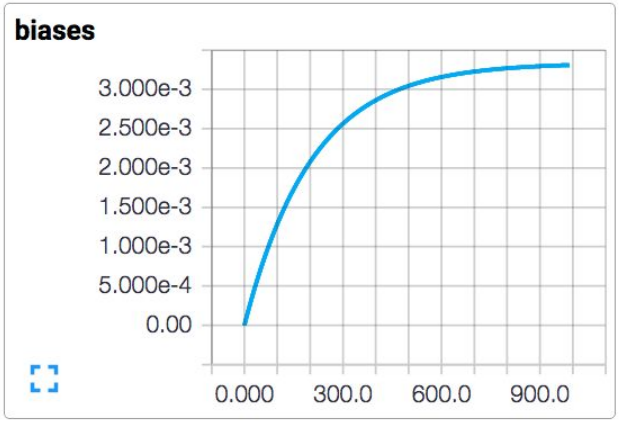
Batch Size = 100, Train Steps = 1000

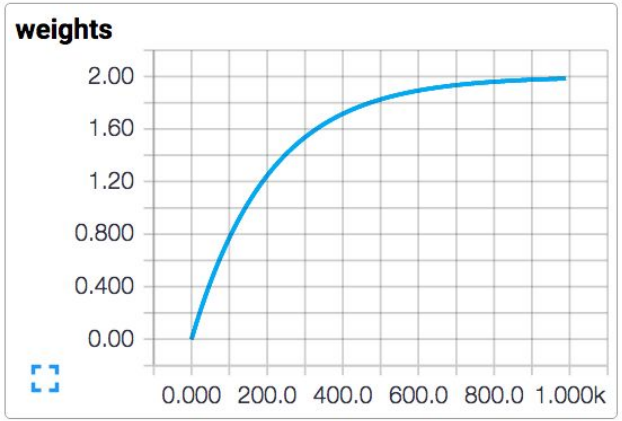


Cost close to 0 = good model fit

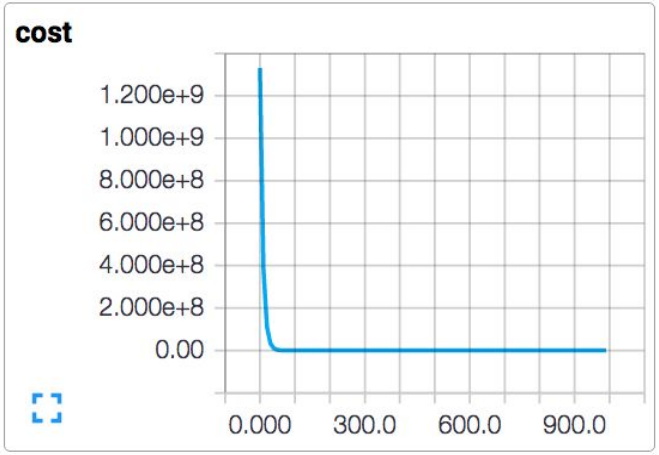
b ~ 0.0

W ~ 2.0





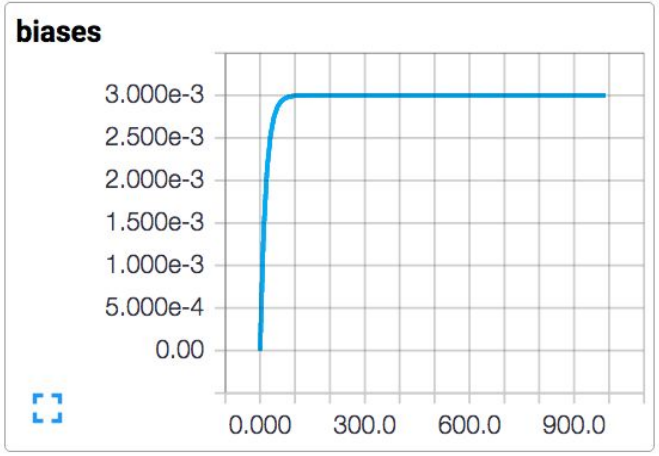
Batch Size = 1000 (Full), Train Steps = 1000

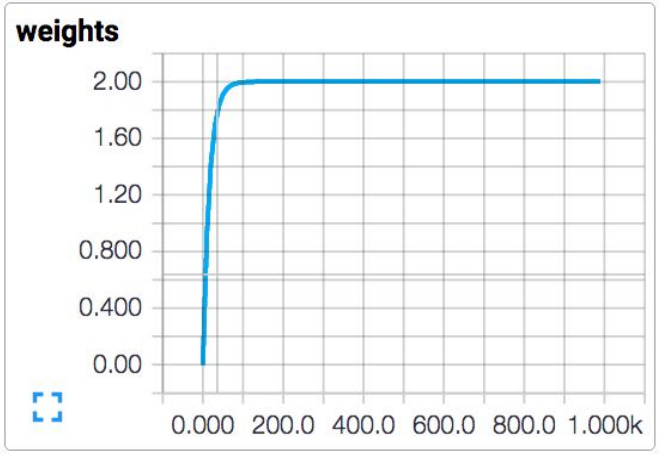


Cost close to 0 = good model fit

b ~ 0.0

W ~ 2.0





Resource to train each step

Less More

[Batch Size 1] [Batch Size 100] [Full Batch]

Many Few

Steps to reach good W, b values

References

Code:

- Using Tensorboard for visualization:
  - https://github.com/nethsix/gentle_tensorflow/blob/master/code/linear_regression_one_feature_with_tensorboard.py
- Performing stochastic/mini-batch/batch Gradient Descent using Tensorflow
  - https://github.com/nethsix/gentle_tensorflow/blob/master/code/linear_regression_one_feature_using_mini_batch_with_tensorboard.py