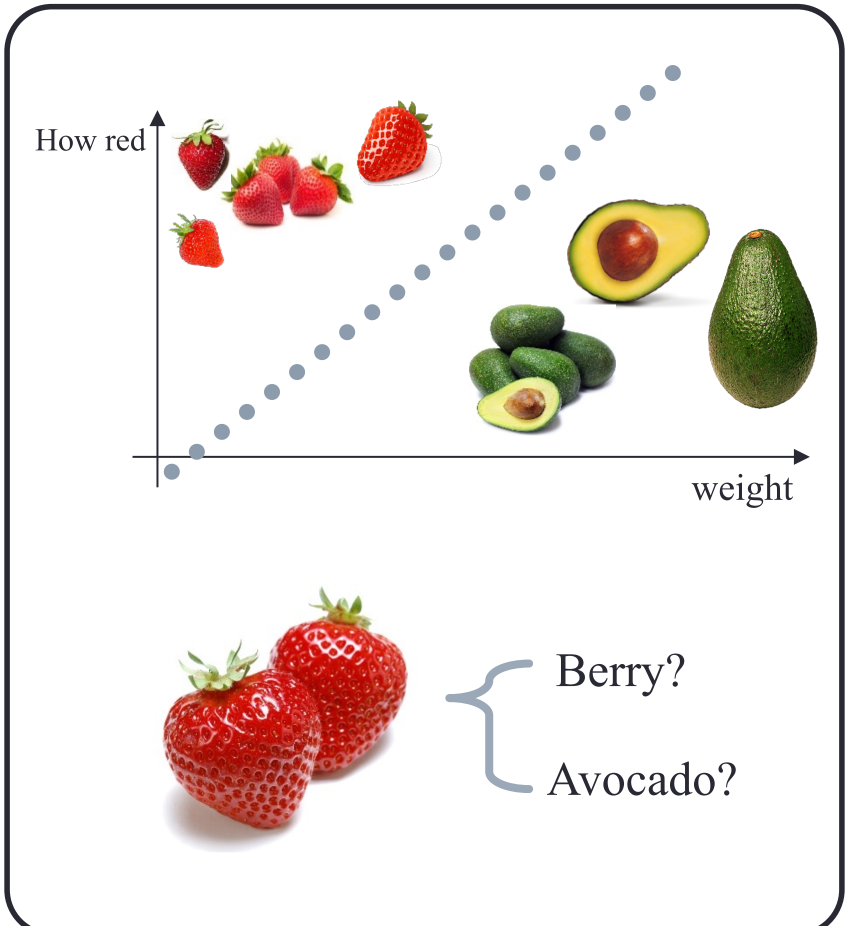
REGRESSION

HYUNG IL KOO

CLASSIFICATION VS REGRESSION

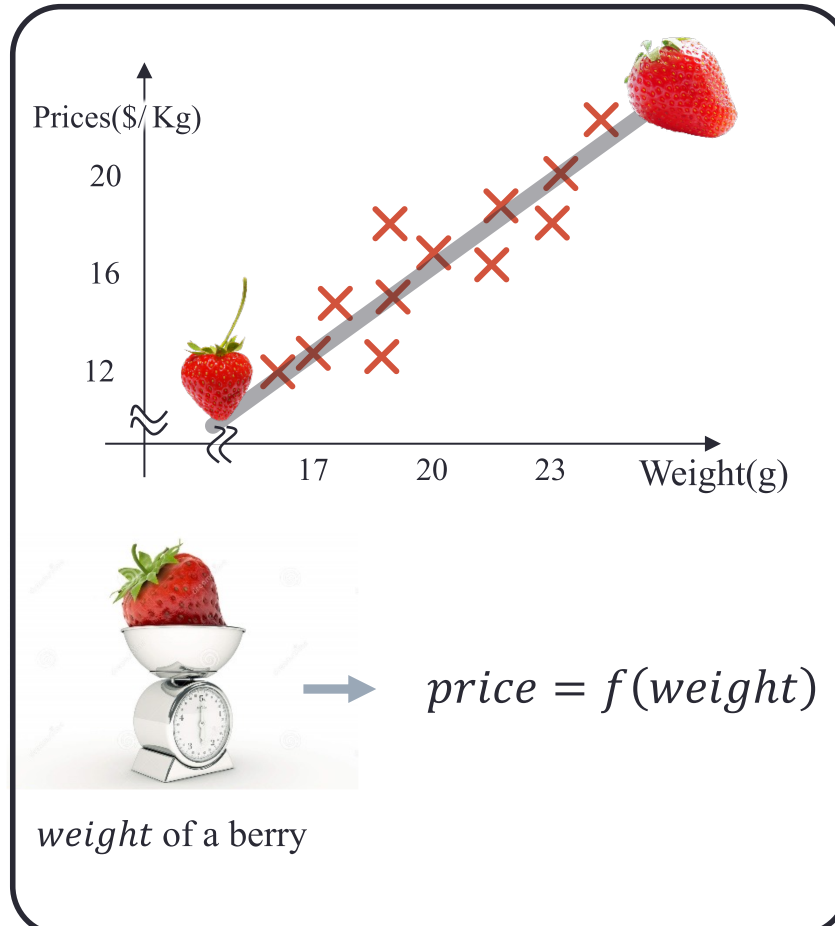
Classification vs Regression

Classification



The variable we are trying to predict is
DISCRETE

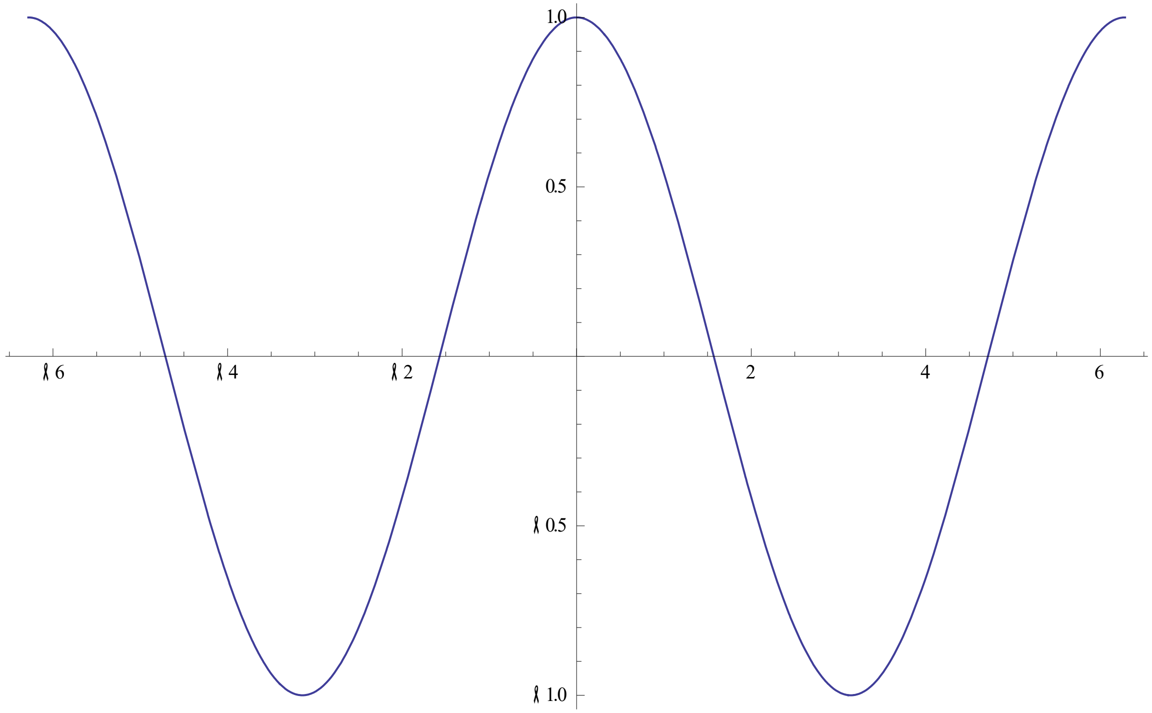
Regression



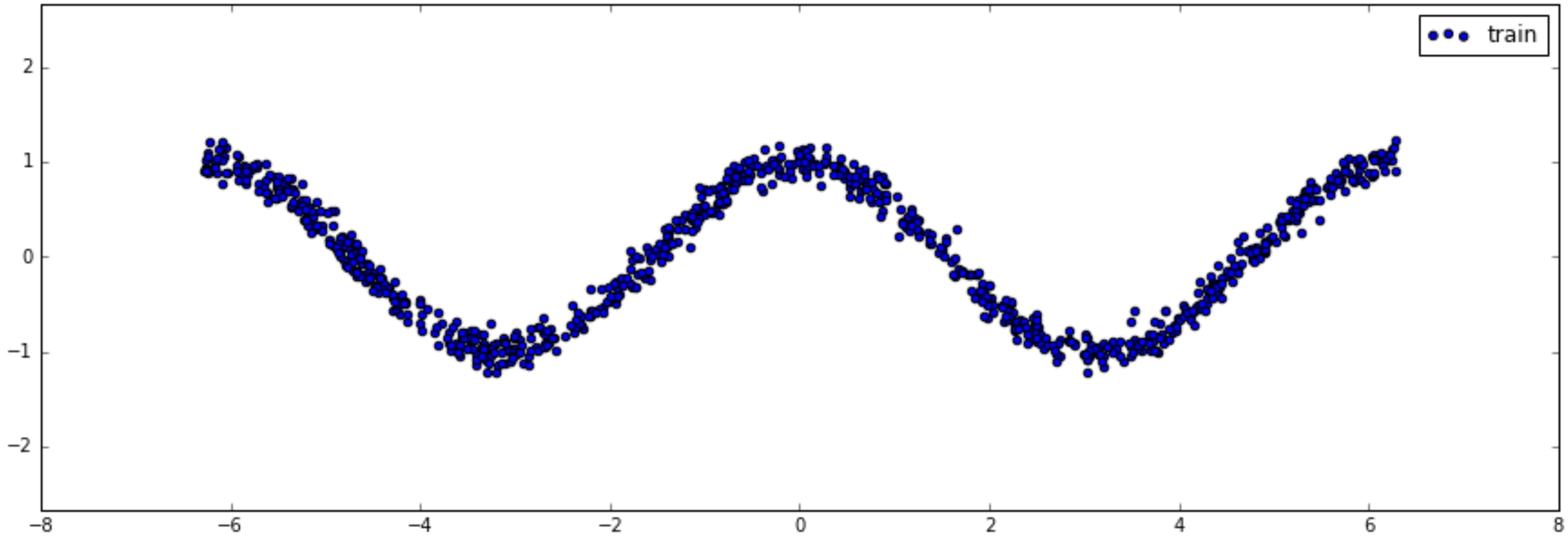
The variable we are trying to predict is
CONTINUOUS

MLP FUNCTION APPROXIMATION

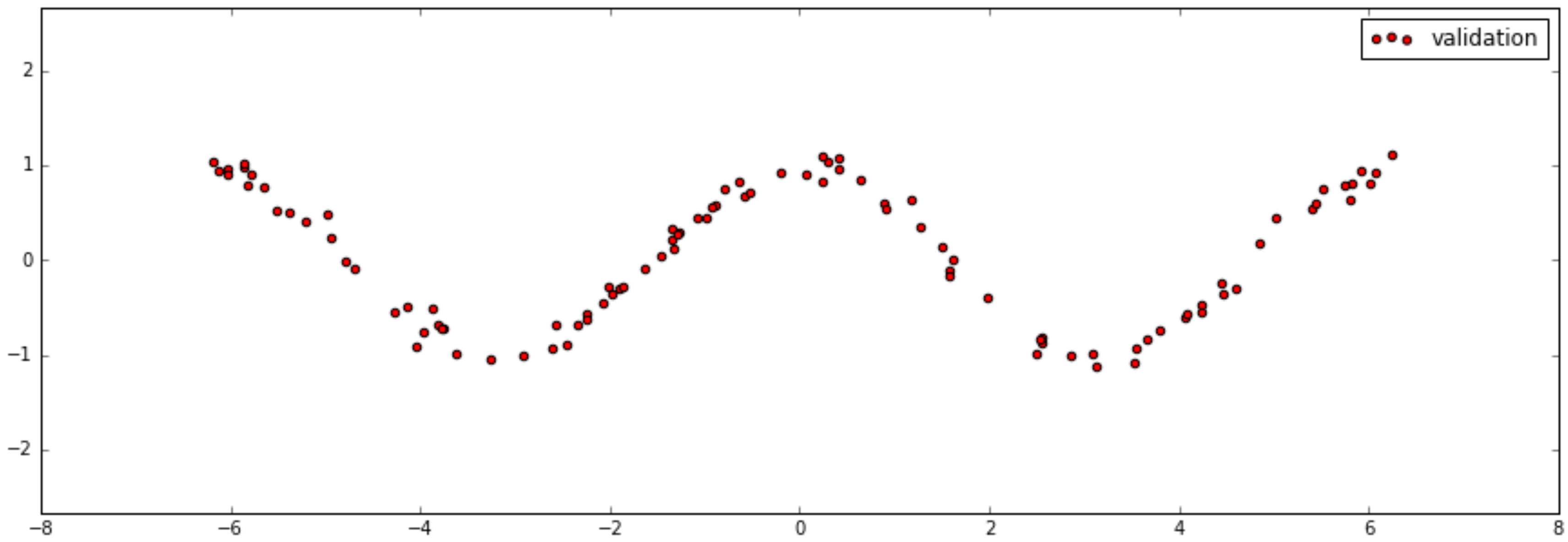
Function Apporoximation



Training samples



Test samples



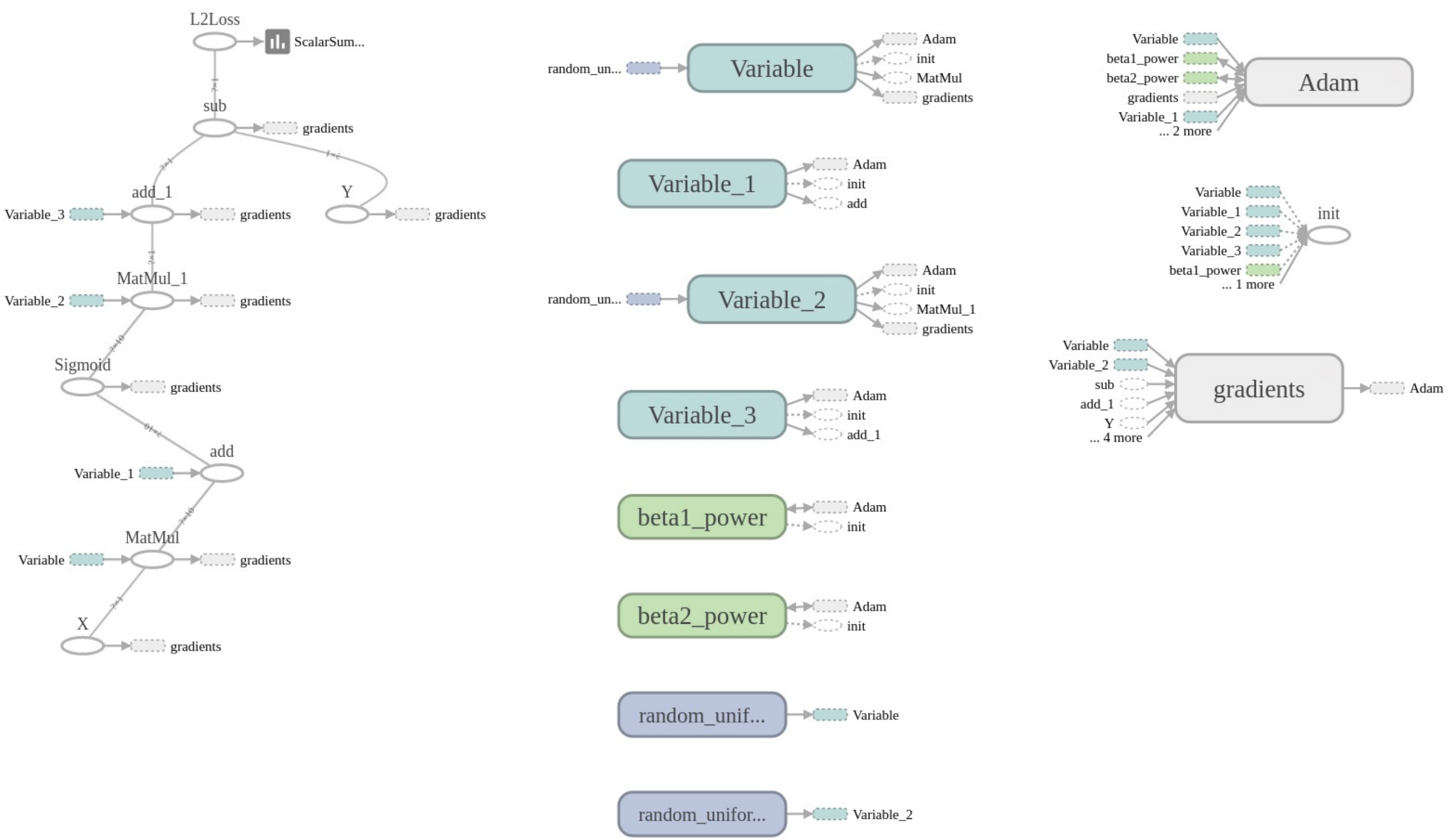
Tensorflow code

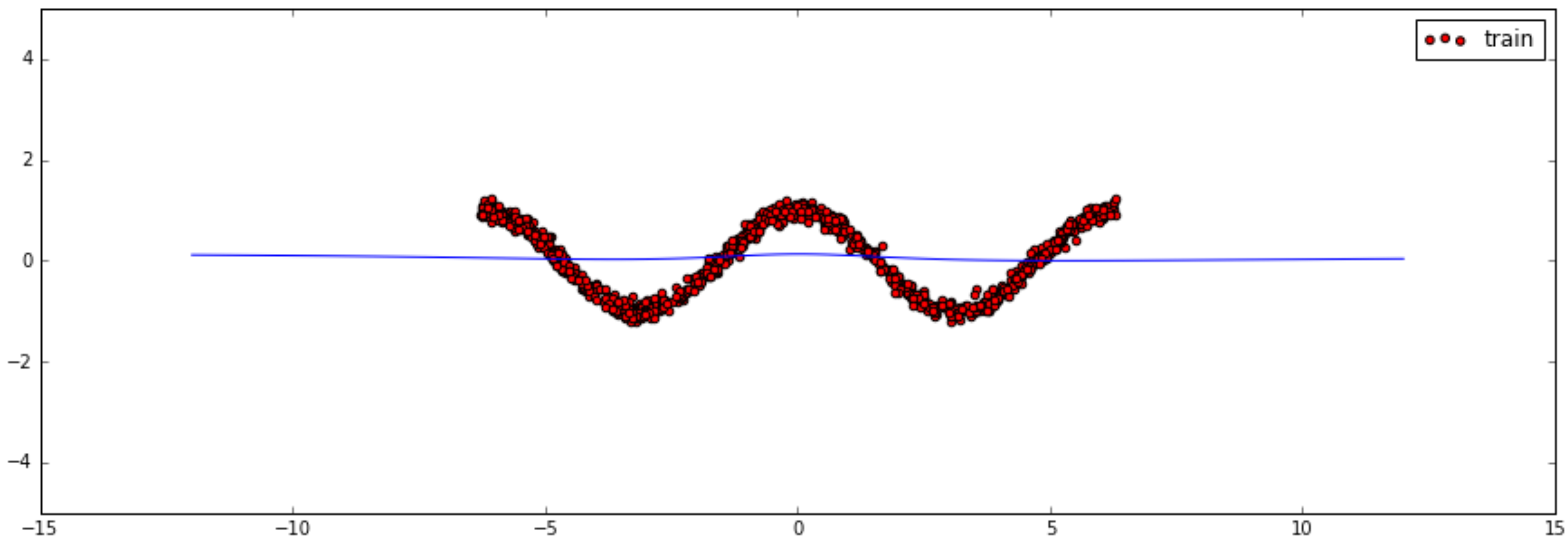
```
import tensorflow as tf
import numpy as np
import math, random
import matplotlib.pyplot as plt


NUM_points = 1000
np.random.seed(NUM_points)
function_to_learn = lambda x: np.cos(x) + 0.1*np.random.randn(*x.shape)

layer_1_neurons = 10

batch_size = 100
NUM_EPOCHS = 1500

all_x = np.float32(np.random.uniform(-2*math.pi,2*math.pi, (1, NUM_points))).T
np.random.shuffle(all_x)
train_size = int(900)


x_training = all_x[:train_size]
y_training = function_to_learn(x_training)

x_validation = all_x[train_size:]
y_validation = function_to_learn(x_validation)


X = tf.placeholder(tf.float32, [None,1], name="X")
Y = tf.placeholder(tf.float32, [None,1], name="Y")
```

Tensorflow code

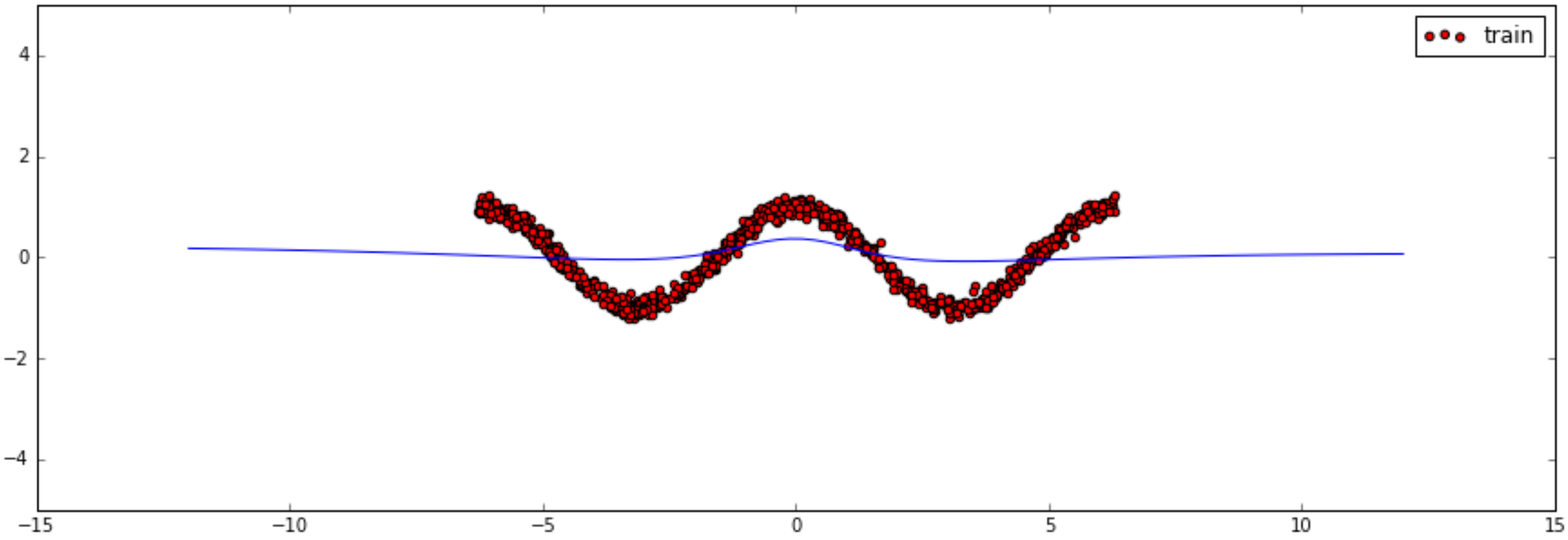
```
w_h = tf.Variable(tf.random_uniform([1, layer_1_neurons], minval=-1,maxval=1,dtype=tf.float32))
b_h = tf.Variable(tf.zeros([1, layer_1_neurons], dtype=tf.float32))

h = tf.nn.sigmoid(tf.matmul(X,w_h)+b_h)

w_o = tf.Variable(tf.random_uniform([layer_1_neurons,1],minval=-1,maxval=1,dtype=tf.float32))
b_o = tf.Variable(tf.zeros([1,1],dtype=tf.float32))

model = tf.matmul(h, w_o) + b_o

train_op = tf.train.GradientDescentOptimizer(0.003).minimize(tf.nn.l2_loss(model-Y))

init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)

train_writer = tf.train.SummaryWriter("./curvefitlog", sess.graph)

errors = []
for i in range(NUM_EPOCHS):
    for start, end in zip( range(0,len(x_training), batch_size), range(batch_size,len(x_training),batch_size)):
        sess.run(train_op, feed_dict={X: x_training[start:end],Y:y_training[start:end]})
    cost = sess.run(tf.nn.l2_loss(model-y_validation),feed_dict = {X:x_validation})
    errors.append(cost)
    if i%100 == 0: print "epoch %d, cost = %g" %(i,cost)
```

Graph

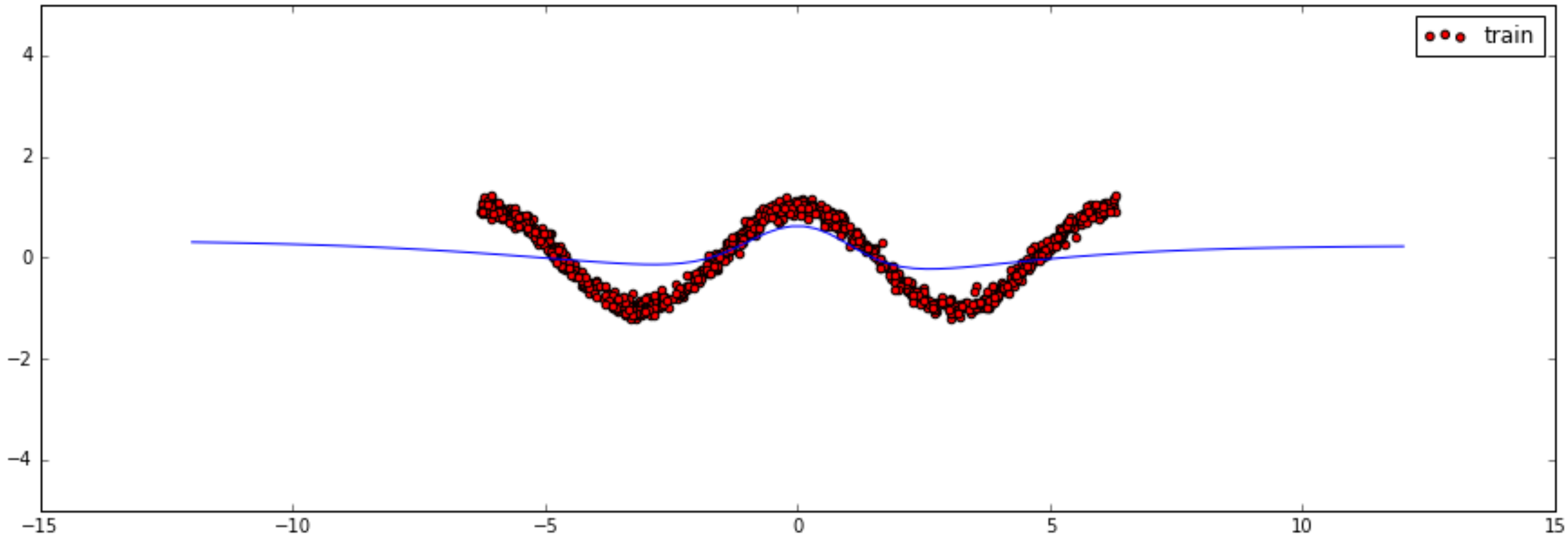

L2Loss
ScalarSum...
sub
gradients
add_1
Y
Variable_3
MatMul_1
Variable_2
Sigmoid
add
Variable_1
MatMul
Variable
X
random_un...
Adam
init
Variable_1
Variable_2
Variable_3
beta1_power
beta2_power
random_unif...
random_unifor...
... 2 more
... 1 more
... 4 more

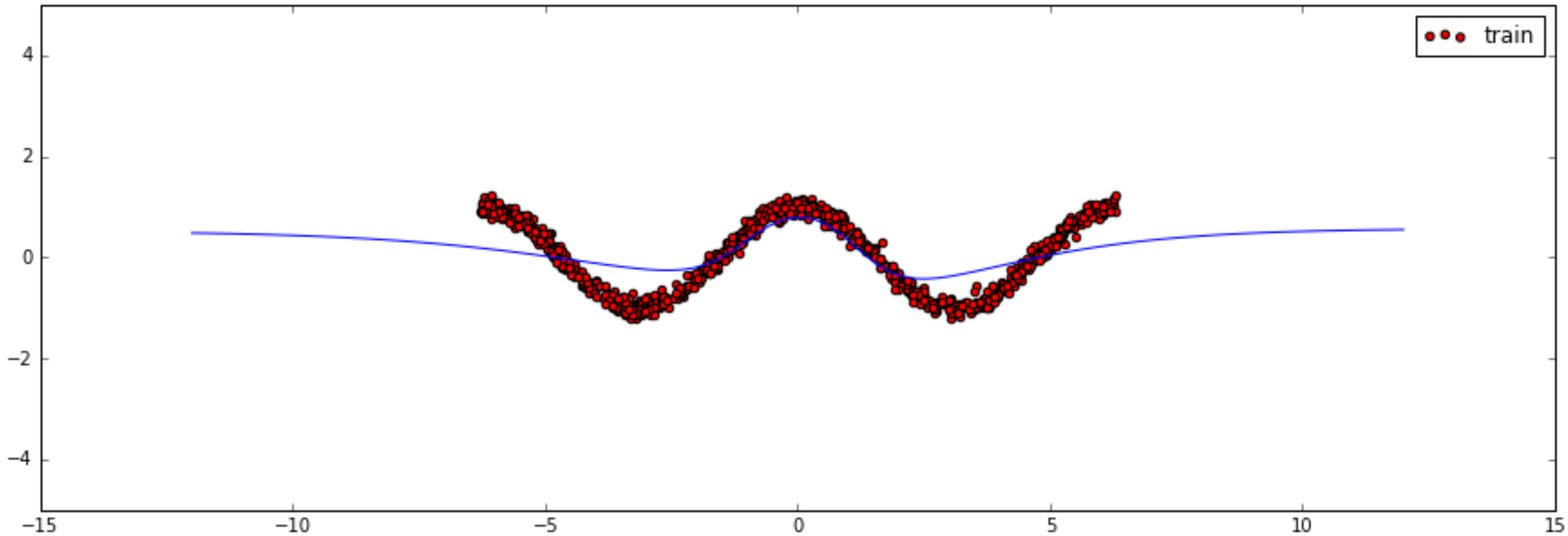


train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15

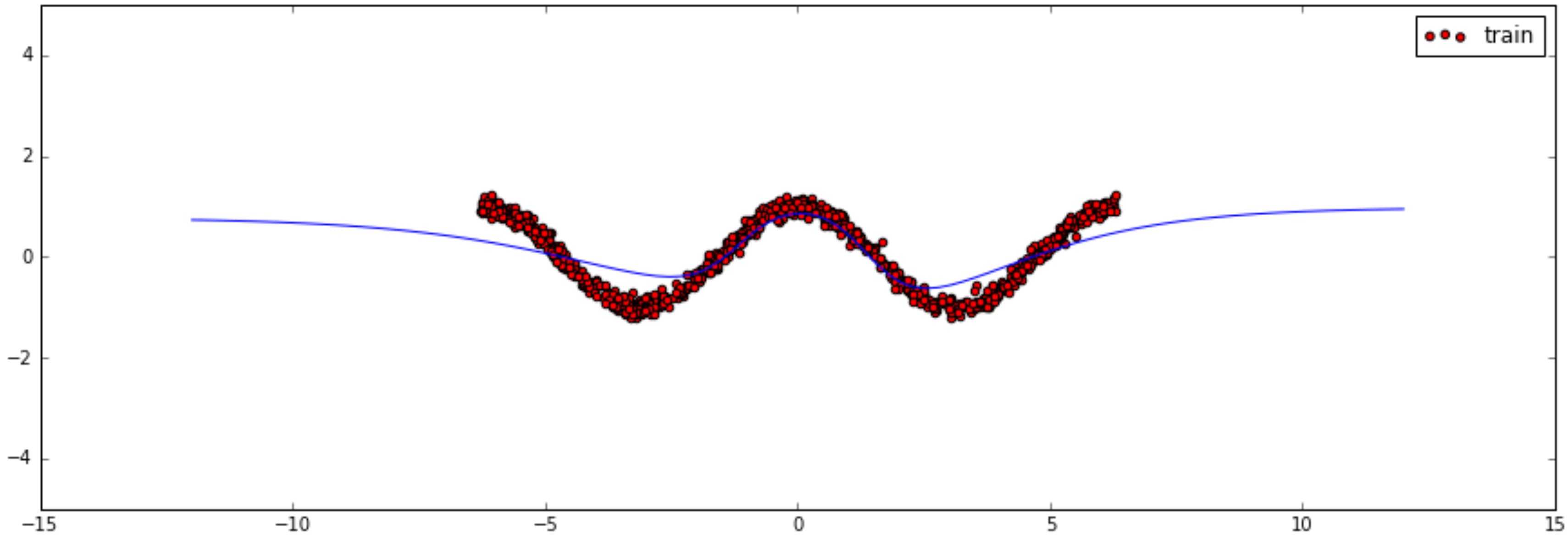


train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15

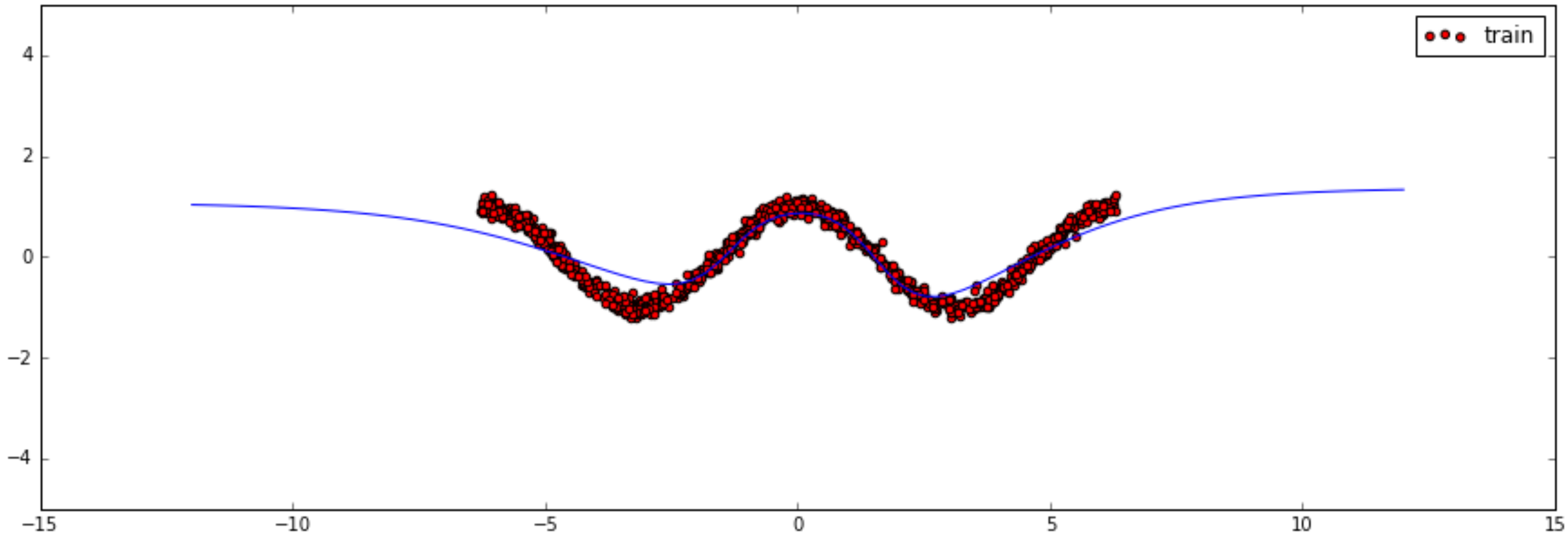

train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15

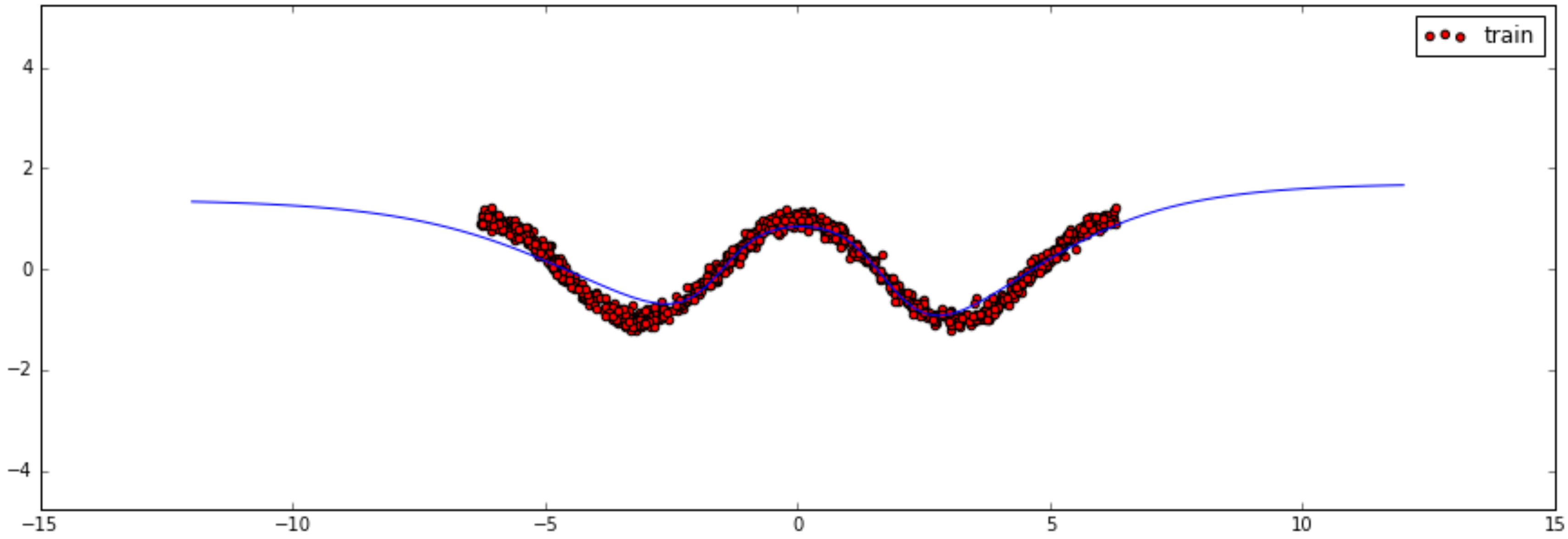


train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15

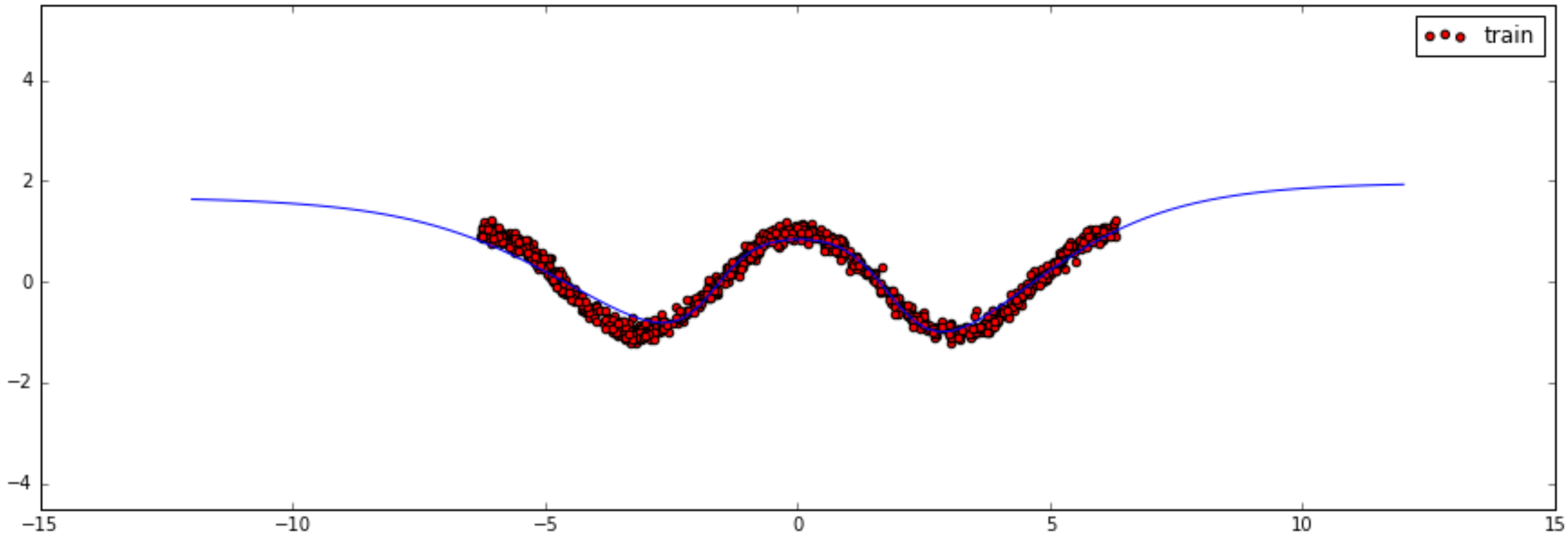


train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15

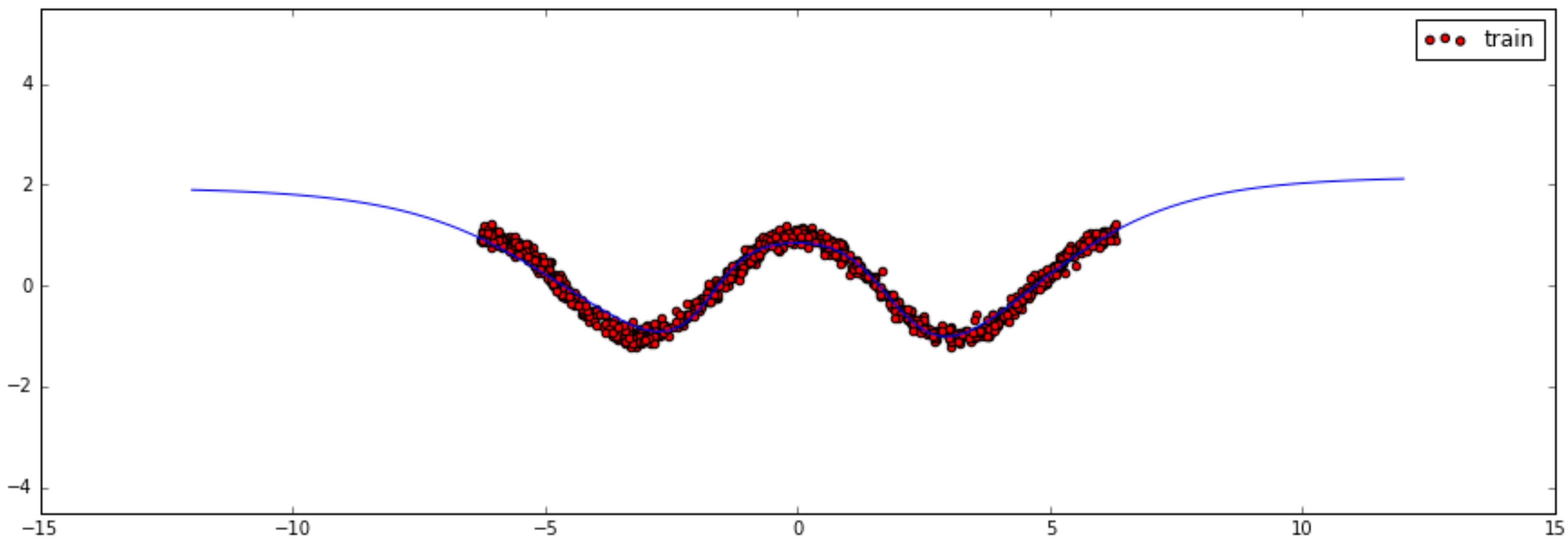

train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15



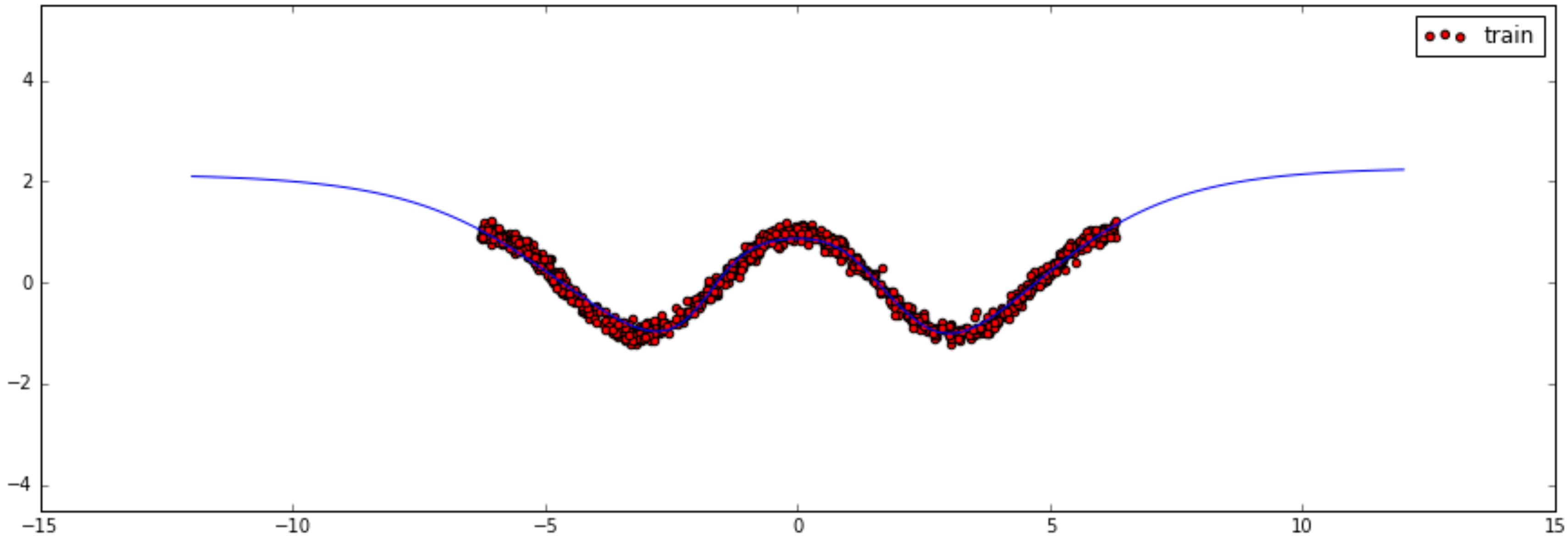
train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15



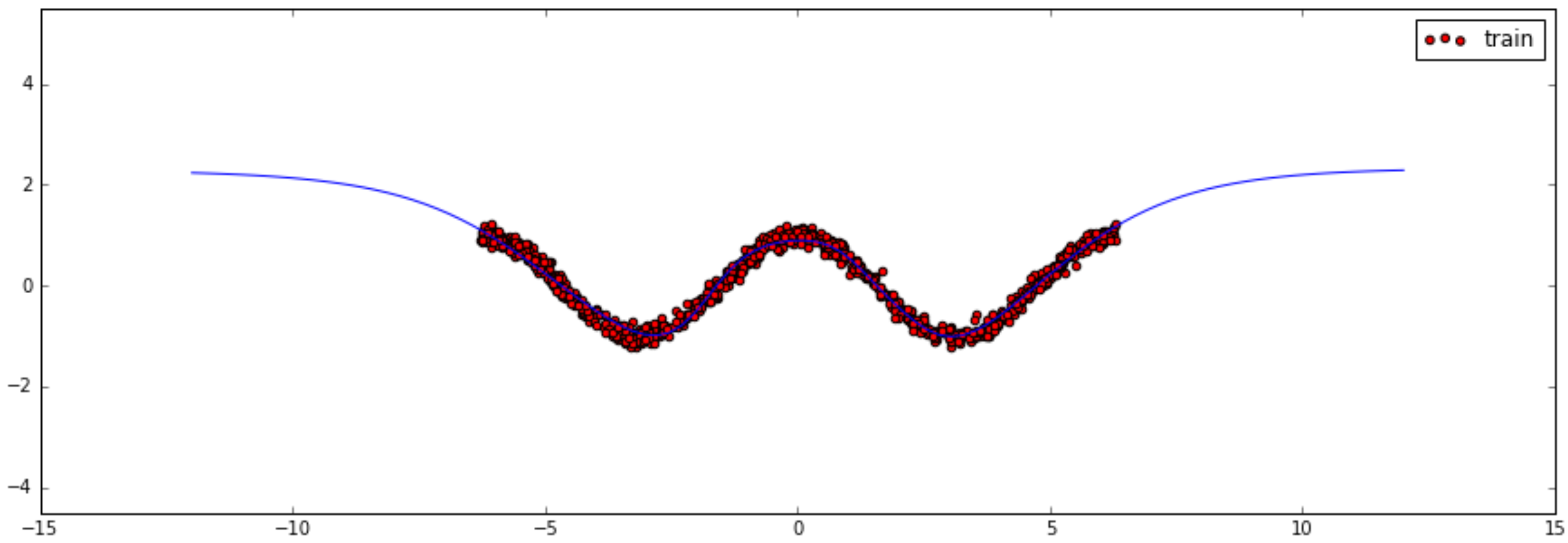
train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15



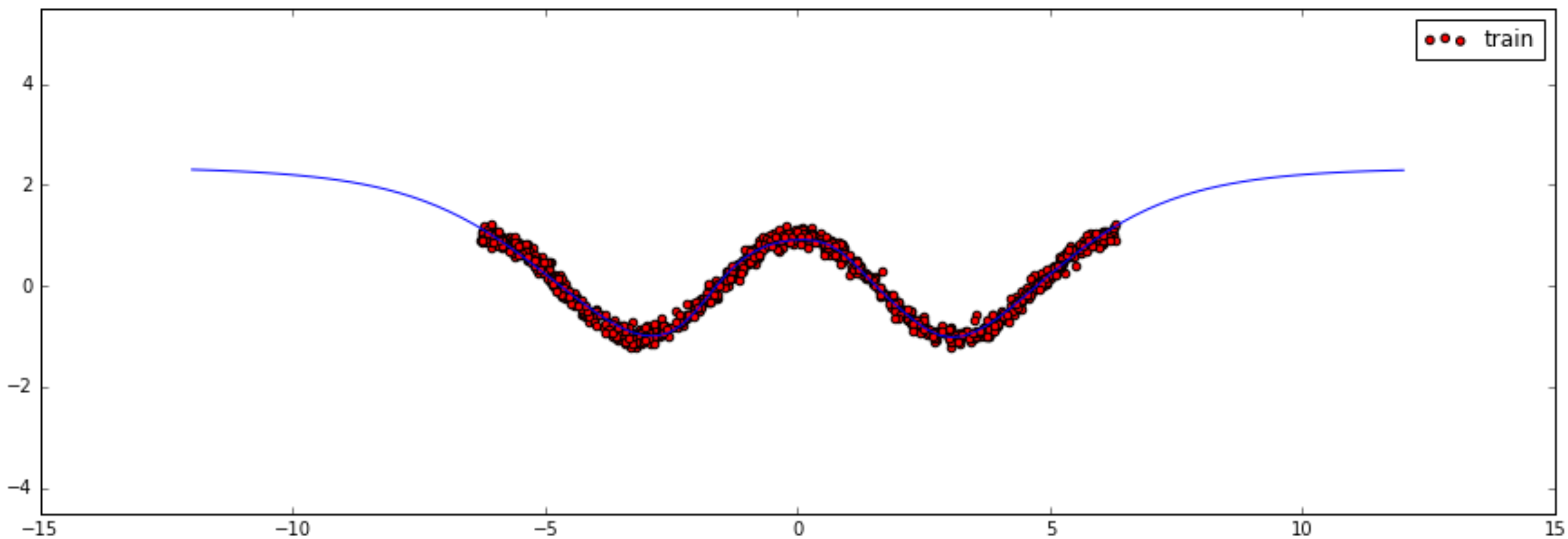
train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15



train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15



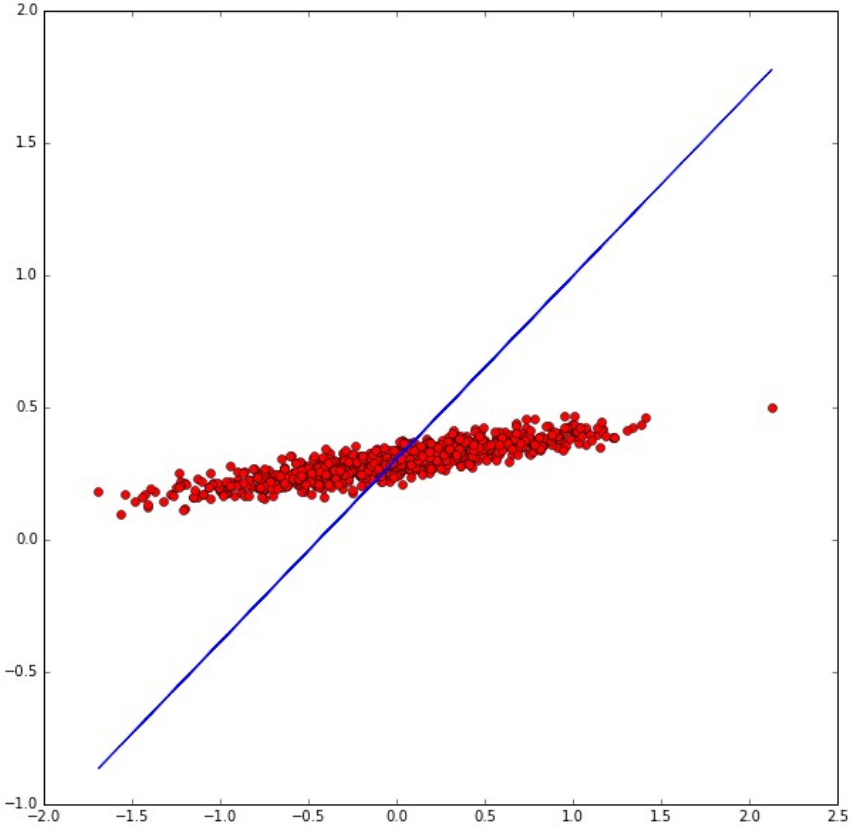
train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15

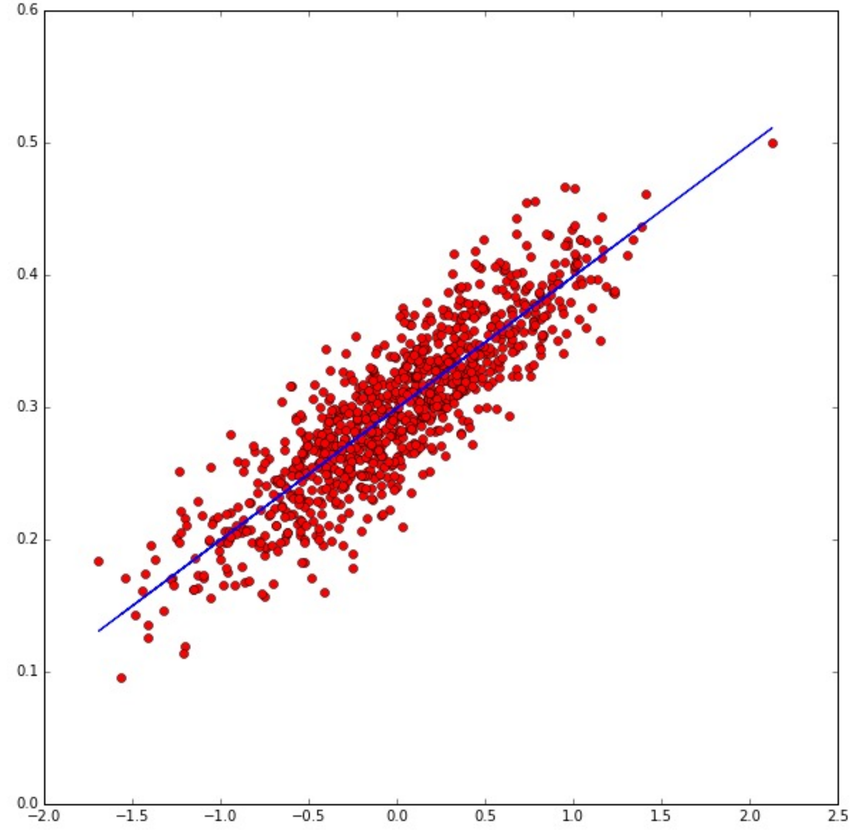


train
4
2
0
−2
−4
−15
−10
−5
0
5
10
15

LINEAR REGRESSION

```python
import tensorflow as tf
import numpy as np

# Create 100 phony x, y data points in NumPy, y = x * 0.1 + 0.3
x_data = np.random.rand(100).astype(np.float32)
y_data = x_data * 0.1 + 0.3

# Try to find values for W and b that compute y_data = W * x_data + b
# (We know that W should be 0.1 and b 0.3, but TensorFlow will
# figure that out for us.)
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W * x_data + b

# Minimize the mean squared errors.
loss = tf.reduce_mean(tf.square(y - y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# Before starting, initialize the variables.  We will 'run' this first.
init = tf.initialize_all_variables()

# Launch the graph.
sess = tf.Session()
sess.run(init)

# Fit the line.
for step in range(201):
    sess.run(train)
    if step % 20 == 0:
        print(step, sess.run(W), sess.run(b))

# Learns best fit is W: [0.1], b: [0.3]
```


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


0.6
0.5
0.4
0.3
0.2
0.1
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5